A Sound Algorithm for Asynchronous Session Subtyping and its Implementation

Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, Gianluigi Zavattaro

Logical Methods in Computer Science
Volume 17, Issue 1, 2021, pp. 20:1–20:35
https://lmcs.episciences.org/

Submitted Dec. 16, 2019
Published Mar. 04, 2021

A SOUND ALGORITHM FOR ASYNCHRONOUS SESSION SUBTYPING AND ITS IMPLEMENTATION

MARIO BRAVETTI[a], MARCO CARBONE[b], JULIEN LANGE[c], NOBUKO YOSHIDA[d], AND GIANLUIGI ZAVATTARO[a]

[a] University of Bologna / INRIA FoCUS Team
e-mail address: {mario.bravetti,gianluigi.zavattaro}@unibo.it

[b] IT University of Copenhagen
e-mail address: carbonem@itu.dk

[c] Royal Holloway, University of London
e-mail address: julien.lange@rhul.ac.uk

[d] Imperial College London
e-mail address: n.yoshida@imperial.ac.uk

Abstract. Session types, types for structuring communication between endpoints in concurrent systems, are recently being integrated into mainstream programming languages. In practice, a very important notion for dealing with such types is that of subtyping, since it allows for typing larger classes of systems, where a program has not precisely the expected behavior but a similar one. Unfortunately, recent work has shown that subtyping for session types in an asynchronous setting is undecidable. To cope with this negative result, the only approaches we are aware of either restrict the syntax of session types or limit communication (by considering forms of bounded asynchrony). Both approaches are too restrictive in practice, hence we proceed differently by presenting an algorithm for checking subtyping which is sound, but not complete (in some cases it terminates without returning a decisive verdict). The algorithm is based on a tree representation of the coinductive definition of asynchronous subtyping; this tree could be infinite, and the algorithm checks for the presence of finite witnesses of infinite successful subtrees. Furthermore, we provide a tool that implements our algorithm. We use this tool to test our algorithm on many examples that cannot be managed with the previous approaches, and to provide an empirical evaluation of the time and space cost of the algorithm.

1. Introduction

Session types are behavioural types that specify the structure of communication between the endpoints of a system or the processes of a concurrent program. In recent years, session types have been integrated into several mainstream programming languages (see, e.g., [HY16, Pad17, SY16, LM16, OY16, ABB^{+}16, NHYA18]) where they specify the pattern of interactions that each endpoint must follow, i.e., a communication protocol. The notion of

Key words and phrases: Session types, and Concurrency, and Subtyping, and Algorithm.

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.23638/LMCS-17(1:20)2021

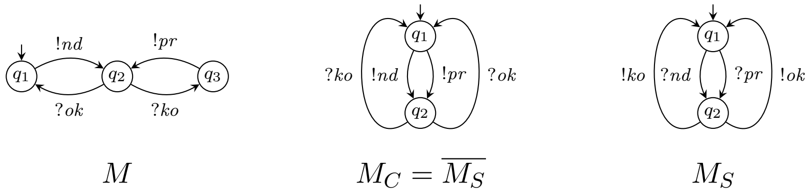


Figure 1: Hospital Service example. M is the (refined) session type of the client, M_C is a supertype of the client M, and M_S is the session type of the server.

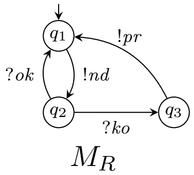


Figure 2: Refined Hospital Service client. M_R is an asynchronous subtype of M_C, i.e., a refined session type of the Hospital Service client.

duality is at the core of theories based on session types, where it guarantees that each send (resp. receive) action is matched by a corresponding receive (resp. send) action, and thus rules out deadlocks [dBBLZ18] and orphan messages. A two-party communication protocol specified as a pair of session types is "correct" (deadlock free, etc) when these types are dual of each other. Unfortunately, in practice, duality is a too strict prerequisite, since it does not provide programmers with the flexibility necessary to build practical implementations of a given protocol. A natural solution for relaxing this rigid constraint is to adopt a notion of (session) subtyping which lets programmers implement refinements of the specification (given as a session type). In particular, an endpoint implemented as program P_2 with type M_2 can always be safely replaced by another program P_1 with type M_1 whenever M_1 is a subtype of M_2 (written $M_1 \preccurlyeq M_2$ in this paper).

The two main known notions of subtyping for session types differ in the type of communication they support: either synchronous (rendez-vous) or asynchronous (over unbounded FIFO channels). *Synchronous session subtyping* checks, by means of a so-called subtyping simulation game, that the subtype implements fewer internal choices (sends), and more external choices (receives), than its supertype. Hence checking whether two types are related can be done efficiently (quadratic time wrt. the size of the types [LY16]). Synchronous session subtyping is of limited interest in modern programming languages such as Go and Rust, which provide *asynchronous* communication over channels. Indeed, in an asynchronous setting, the programmer needs to be able to make the best of the flexibility given by non-blocking send actions. This is precisely what the *asynchronous session subtyping* offers: it widens the synchronous subtyping relation by allowing the subtype to anticipate send (output) actions, when this does not affect its communication partner, i.e., it will notably execute all required receive (input) actions later.

We illustrate the salient points of the asynchronous session subtyping with Figures 1 and 2, which depict the hypothetical session types of the client and server endpoints of a Hospital Service, represented as communicating machines — an equivalent formalism [BZ83, DY12], see Figure 3. Let us consider Figure 1 first. Machine M_S (right) is a server which can deal with two types of requests: it can receive either a message *nd* (next patient data) or a message *pr* (patient report). After receiving a message of either type, the server replies with *ok* or *ko*, indicating whether the evaluation of received data was successful or not, then it returns to its starting state. Machine M_C (middle) represents the type of the client. It is the *dual* of the server M_S (written $\overline{M_S}$), as required in standard two-party session types without subtyping. A programmer may want to implement a slightly improved program which behaves as Machine M (left). This version starts by sending *nd*, then keeps sending patient reports (*pr*) until the previously sent data are deemed satisfactory (it receives *ok*). In fact, machine M is a *synchronous* subtype of machine M_C, because of the covariance of outputs, i.e., M is a subtype of M_C, hence it can send fewer messages. Note that M can receive the same messages as M_C. Machine M_R in Figure 2 is another refinement of Machine M_C, but M_R is not a synchronous subtype of M_C. Instead, M_R is an *asynchronous* subtype of M_C. Indeed, M_R is able to receive the same set of messages as M_C, each of the sent messages are also allowed by M_C, and the system consisting of the parallel composition of machines M_R and M_S communicating via unbounded FIFO channels is free from deadlocks and orphan messages. We will use this example ($M_R \preccurlyeq M_C$) in the rest of the paper to illustrate our theory. Figure 3 gives the session types corresponding to the machines in Figures 1 and 2, where & indicates an external choice and $\oplus$ indicates an internal choice.

Recently, we have proven that checking whether two types are in the asynchronous subtyping relation is, unfortunately, *undecidable* [BZ20, BCZ18, LY17, BCZ17]. In order to mitigate this negative result, some theoretical algorithms have been proposed for restricted subclasses of session types. These restrictions can be divided into two main categories: syntactical restrictions, i.e., allowing only one type of non-unary branching (internal or external choice), or adding bounds on the number of pending messages in FIFO communication channels. Both types of restrictions are problematic in practice. Syntactic restrictions disallow protocols featuring both types of internal/external choices, e.g., the machines M_C and M_S in Figure 1 contain (non-unary) external and internal choices. On the other hand, applying a bound to the subtyping relation is generally difficult because (*i*) it is generally undecidable whether such a bound exists, (*ii*) the channel bounds used in the implementation (if any) might not be known at compile time, and (*iii*) very simple systems, such as the one consisting of the parallel composition of machines M_R and M_S discussed above, require unbounded communication channels.

The main contribution of this paper is to give a sound algorithm for checking asynchronous session subtyping that does not impose syntactical restrictions nor bounds as done in previous works.

Overview of our approach. Our approach will allow to algorithmically check the subtyping between session types like M_R and M_C. In a nutshell, our algorithm proceeds as follows. We play the classical subtyping simulation game with the subtype and supertype candidates. The game terminates when we encounter a *failure*, meaning that the two types are not in the subtyping relation, or when we detect a *repetitive* behaviour in the game. In the latter case, we check whether this repetitive behaviour (which can always be found) satisfies sufficient conditions that guarantee that the subtyping simulation game will never encounter failures. If the conditions are satisfied the algorithm concludes that the two types are in the subtyping relation, otherwise no final verdict is returned.

More precisely, session subtyping is defined following a coinductive approach (Definition 2.6) that formalises a check on the types that can be intuitively seen as a game. At each step of the game, the candidate subtype proposes a challenge (either an input or an output action to be executed) and the candidate supertype is expected to reply by performing a corresponding action. The game ends in two possible ways: either both types terminate by reaching their end state (*success*) or the candidate supertype is unable to reply to the challenge (*failure*). In case of failure, the two types are not in the subtyping relation, otherwise they are. This game is the so-called *subtyping simulation game*, and we formally represent it as a *simulation tree* (Definition 3.2). Hence two types are in the subtying relation if and only if their simulation tree does not reach a failure (Theorem 3.4).

Recall that asynchronous session subtyping allows the subtype to anticipate output actions wrt. the supertype. Hence, during the subtyping simulation game, a supertype can reply to an output challenge by considering outputs that are not immediately available, but are guarded by inputs. These inputs cannot be forgotten during the game, because they could be necessary to reply to subsequent input challenges. Thus, they are recorded in so-called *input trees* (Definition 2.2). Due to outputs inside loops, we can accumulate an unbounded amount of inputs, thus generating input trees of unbounded depth. For this reason, it is generally not possible to algorithmically compute the entire simulation tree. To overcome this problem, we propose a termination condition that intuitively says that the computation of the simulation tree can be stopped when we reach a point in the game that precisely corresponds to a previous point, or differs simply because "more" inputs have been accumulated (Theorem 3.8).

Using this termination condition, we compute a finite prefix of the simulation tree. Given this finite tree, our algorithm proceeds as follows: (*i*) it extracts special subtrees, called *candidate subtrees*, from the tree (Definition 3.6), and then (*ii*) checks whether all these subtrees satisfy certain properties guaranteeing that, even if we have stopped the game, it would certainly continue without reaching a failure. This is guaranteed if we have stopped the computation of the simulation tree by reaching an already considered point, because subsequent continuations of the game will continue repeating the exact same steps. In contrast, if we have stopped with "more" inputs, we must have the guarantee that all possible continuations of the simulation game cannot be negatively affected by these additional input accumulations. We formalise a sufficient condition on candidate subtrees (that are named *witness trees* when they satisfy such a condition, see Definition 3.16) that provides such a guarantee.

Concretely we use input tree equations (a sort of context-free tree grammar, see Definition 3.11) to finitely represent both the possible inputs of the candidate subtype and the inputs that can be accumulated by the candidate supertype. We then define a *compatibility* relation on input tree equations, see Definition 3.12. In a witness tree we impose that the input tree equations of the inputs accumulated by the candidate supertype are compatible with those of the candidate subtype. This implies that the candidate supertype will be always ready to reply to all possible input challenges of the candidate subtype, simply by considering already accumulated inputs (see our main Theorem 3.19). If all the candidate subtrees satisfy our sufficient conditions we can conclude that the two initial session types are in the subtyping relation, otherwise the algorithm replies with "I don't know" meaning that it is not possible to conclude with a final verdict.

1.1. **Structure of the paper.**

The remainder of the paper is structured as follows. § 2 reports some preliminary definitions, namely the formalisation of session types as communicating machines and the definition of asynchronous session subtyping. Our approach for a sound algorithmic characterisation of asynchronous session subtyping is presented in § 3.

$$
\begin{aligned}
M &= \mu\mathbf{x}.\oplus\Big\{nd\colon\ \mu\mathbf{y}.\&\{\ ok\colon \mathbf{x},\ \ ko\colon \oplus\{pr\colon \mathbf{y}\}\ \}\ \Big\}\\
M_R &= \mu\mathbf{x}.\oplus\Big\{nd\colon \&\{\ ok\colon \mathbf{x},\ \ ko\colon \oplus\{pr\colon \mathbf{x}\}\ \}\ \Big\}\\
M_C &= \mu\mathbf{x}.\oplus\big\{nd\colon \&\{\ ok\colon \mathbf{x},\ \ ko\colon \mathbf{x}\ \},\quad pr\colon \&\{\ ok\colon \mathbf{x},\ \ ko\colon \mathbf{x}\ \}\ \big\}\\
M_S &= \mu\mathbf{x}.\&\big\{nd\colon \oplus\{ok\colon \mathbf{x},\ ko\colon \mathbf{x}\},\quad pr\colon \oplus\{ok\colon \mathbf{x},\ ko\colon \mathbf{x}\}\big\}
\end{aligned}
$$

Figure 3: Session types corresponding to the machines in Figures 1 and 2.

We also discuss in § 4 a full implementation of our algorithm; this has been used to test our approach on many examples that cannot be managed with the previous approaches, and to provide an empirical evaluation of the time and space cost of the algorithm. Finally, the paper includes a discussion about related work in § 5 and some concluding remarks in § 6.

This article is a full version of [BCL$^+$19a], with improved presentation, refined definitions, detailed proofs and additional examples. Moreover, this version presents an empirical evaluation of our algorithm: we tested the implementation of our algorithm on automatically generated session types, see § 4. We have also given an expanded discussion of related work and possible extensions that can be addressed in the future, see § 5 and § 6.

2. Communicating Machines and Asynchronous Subtyping

In this section, we recall the definition of two-party communicating machines, that communicate over unbounded FIFO channels (§ 2.1), and define asynchronous subtyping for session types [CDSY17, CDCY14], which we adapt to communicating machines, following [BCZ18] (§ 2.2).

2.1. Communicating Machines.

Let $\mathbb{A}$ be a (finite) alphabet, ranged over by a, b, etc. We let ω, ω', etc. range over words in $\mathbb{A}^*$. The set of send (resp. receive) actions is $Act_! = \{!\} \times \mathbb{A}$, (resp. $Act_? = \{?\} \times \mathbb{A}$). The set of actions is $Act = Act_! \cup Act_?$, ranged over by ℓ, where a send action $!a$ puts message a on an (unbounded) buffer, while a receive action $?a$ represents the consumption of a from a buffer. We define $dir(!a) \overset{\text{def}}{=} !$ and $dir(?a) \overset{\text{def}}{=} ?$ and let ψ and φ range over Act^*. We write $\cdot$ for the concatenation operator on words and we write ϵ for the empty word (overloaded for $\mathbb{A}$ and $\mathbb{A}^*$).

In this work, we only consider communicating machines which correspond to (two-party) session types. Hence, we focus on deterministic (communicating) finite-state machines, without mixed states (i.e., states that can fire both send and receive actions) as in [DY12, DY13].

Definition 2.1 (Communicating Machine)**.** A communicating machine M is a tuple (Q, q_0, δ) where Q is the (finite) set of states, $q_0 \in Q$ is the initial state, and $\delta \in Q \times Act \times Q$ is a transition relation. We further require that $\forall q, q', q'' \in Q.\ \forall \ell, \ell' \in Act$:

(1) $(q, \ell, q'), (q, \ell', q'') \in \delta$ implies $dir(\ell) = dir(\ell')$, and
(2) $(q, \ell, q'), (q, \ell, q'') \in \delta$ implies $q' = q''$.

We write $q \xrightarrow{\ell} q'$ for $(q, \ell, q') \in \delta$, omit unnecessary labels, and write $\rightarrow^*$ for the reflexive transitive closure of $\rightarrow$.

Condition (1) requires all states to be directed, while Condition (2) enforces determinism, i.e., all actions outgoing from a given state are pairwise distinct.

Given $M = (Q, q_0, \delta)$, we say that $q \in Q$ is *final*, written $q \not\rightarrow$, iff $\forall q' \in Q.\ \forall \ell \in \mathit{Act}.\ (q, \ell, q') \notin \delta$. A state $q \in Q$ is *sending* (resp. *receiving*) iff q is not final and $\forall q' \in Q.\ \forall \ell \in \mathit{Act}.\ (q, \ell, q') \in \delta.\ \mathit{dir}(\ell) = !$ (resp. $\mathit{dir}(\ell) = ?$). We use $\delta(q, \ell)$ to stand for q' such that $(q, \ell, q') \in \delta$.

We write $q_0 \xrightarrow{\ell_1 \cdots \ell_k} q_k$ iff there are $q_1, \ldots, q_{k-1} \in Q$ such that $q_{i-1} \xrightarrow{\ell_i} q_i$ for $1 \leq i \leq k$. Given a list of messages $\omega = a_1 \cdots a_k$ $(k \geq 0)$, we write $?\omega$ for the list $?a_1 \cdots ?a_k$ and $!\omega$ for $!a_1 \cdots !a_k$.

Given $\psi \in \mathit{Act}^*$ we define $\mathsf{snd}(\psi)$ and $\mathsf{rcv}(\psi)$:

$$\mathsf{snd}(\psi) = \begin{cases} a \cdot \mathsf{snd}(\psi') & \text{if } \psi = !a \cdot \psi' \\ \mathsf{snd}(\psi') & \text{if } \psi = ?a \cdot \psi' \\ \epsilon & \text{if } \psi = \epsilon \end{cases} \qquad \mathsf{rcv}(\psi) = \begin{cases} a \cdot \mathsf{rcv}(\psi') & \text{if } \psi = ?a \cdot \psi' \\ \mathsf{rcv}(\psi') & \text{if } \psi = !a \cdot \psi' \\ \epsilon & \text{if } \psi = \epsilon \end{cases}$$

That is $\mathsf{snd}(\psi)$ (resp. $\mathsf{rcv}(\psi)$) extracts the messages in send (resp. receive) actions from a sequence ψ.

2.2. **Asynchronous Session Subtyping.**

2.2.1. *Input trees and contexts.* We define some structures and functions which we use to formalise the subtyping relation. In particular, we use syntactic constructs used to record the input actions that have been anticipated by a candidate supertype, e.g., machine M_2 in Definition 2.6, as well as the local states it may reach. First, input trees (Definition 2.2) record input actions in a standard tree structure.

Definition 2.2 (Input Tree)**.** An input tree is a term of the grammar:

$$T \quad ::= \quad q \ \mid \ \langle a_i : T_i \rangle_{i \in I}$$

In the sequel, we use $\mathcal{T}_Q$ to denote the input trees over states $q \in Q$. An input context is an input tree with "holes" in the place of sub-terms.

Definition 2.3 (Input Context)**.** An input context is a term of $\mathcal{A} \ ::= \ [\,]_j \ \mid \ \langle a_i : \mathcal{A}_i \rangle_{i \in I}$, where all indices j, denoted by $I(\mathcal{A})$, are distinct and are associated to holes.

For input trees and contexts of the form $\langle a_i : T_i \rangle_{i \in I}$ and $\langle a_i : \mathcal{A}_i \rangle_{i \in I}$, we assume that $I \neq \emptyset$, $\forall i \neq j \in I.\ a_i \neq a_j$, and that the order of the sub-terms is irrelevant. When convenient, we use set-builder notation to construct input trees or contexts, e.g., $\langle a_i : T_i \mid i \in I \rangle$.

Given an input context $\mathcal{A}$ and an input context $\mathcal{A}_i$ for each i in $I(\mathcal{A})$, we write $\mathcal{A}[\mathcal{A}_i]^{i \in I(\mathcal{A})}$ for the input context obtained by replacing each hole $[\,]_i$ in $\mathcal{A}$ by the input context $\mathcal{A}_i$. We write $\mathcal{A}[T_i]^{i \in I(\mathcal{A})}$ for the input tree where holes are replaced by input trees.

2.2.2. *Auxiliary functions.* In the rest of the paper we use the following auxiliary functions on communicating machines. Given a machine $M = (Q, q_0, \delta)$ and a state $q \in Q$, we define:

- $\mathsf{cycle}(\star, q) \iff \exists \omega \in \mathbb{A}^*, \omega' \in \mathbb{A}^+, q' \in Q.\ q \xrightarrow{\star\omega} q' \xrightarrow{\star\omega'} q'$ (with $\star \in \{!, ?\}$),
- $\mathsf{in}(q) = \{a \mid \exists q'.q \xrightarrow{?a} q'\}$ and $\mathsf{out}(q) = \{a \mid \exists q'.q \xrightarrow{!a} q'\}$,
- let the *partial* function $\mathsf{inTree}(\cdot)$ be defined as:

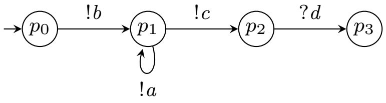
$$\mathsf{inTree}(q) = \begin{cases} \perp & \text{if } \mathsf{cycle}(?, q) \\ q & \text{if } \mathsf{in}(q) = \varnothing \\ \langle a_i : \mathsf{inTree}(\delta(q, ?a_i))\rangle_{i\in I} & \text{if } \mathsf{in}(q) = \{a_i \mid i \in I\} \neq \varnothing \end{cases}$$

Predicate $\mathsf{cycle}(\star, q)$ says that, from q, we can reach a cycle with only sends (resp. receives), depending on whether $\star =!$ or $\star =?$. The function $\mathsf{in}(q)$ (resp. $\mathsf{out}(q)$) returns the messages that can be received (resp. sent) from q. When defined, $\mathsf{inTree}(q)$ returns the tree containing all sequences of messages which can be received from q until a final or sending state is reached. Intuitively, $\mathsf{inTree}(q)$ is undefined when $\mathsf{cycle}(?, q)$ as it would return an infinite tree.

Example 2.4. Given M_C (Figure 1), we have the following:

$$\begin{array}{ll} \mathsf{in}(q_1) = \emptyset & \mathsf{in}(q_2) = \{ok, ko\} \\ \mathsf{out}(q_1) = \{nd, pr\} & \mathsf{out}(q_2) = \emptyset \\ \mathsf{inTree}(q_1) = q_1 & \mathsf{inTree}(q_2) = \langle ok : q_1,\ ko : q_1\rangle \end{array}$$

Example 2.5. Consider the following machine M_1:



From state p_0 we can reach state p_1 with an output. The latter can loop into itself. Hence, we have both $\mathsf{cycle}(!, p_0)$ and $\mathsf{cycle}(!, p_1)$.

2.2.3. *Asynchronous subtyping.* We present our definition of asynchronous subtyping (following the orphan-message-free version from [CDCY14]). Our definition is a simple adaptation[1] of [BCZ18, Definition 2.4] (given on syntactical session types) to the setting of communicating machines.

Definition 2.6 (Asynchronous Subtyping)**.** Let $M_i = (Q_i, q_{0_i}, \delta_i)$ for $i \in \{1, 2\}$. $\mathcal{R}$ is an asynchronous subtyping relation on $Q_1 \times \mathcal{T}_{Q_2}$ such that $(p, T) \in \mathcal{R}$ implies:

(1) if $p \nrightarrow$ then $T = q$ such that $q \nrightarrow$;
(2) if p is a receiving state then
  (a) if $T = q$ then q is a receiving state and
  $\forall q' \in Q_2\ s.t.\ \ q \xrightarrow{?a} q'.\ \exists p' \in Q_1\ s.t.\ p \xrightarrow{?a} p' \wedge (p', q') \in \mathcal{R}$;
  (b) if $T = \langle a_i : T_i\rangle_{i\in I}$ then $\forall i \in I.\ \exists p' \in Q_1\ s.t.\ p \xrightarrow{?a_i} p' \wedge (p', T_i) \in \mathcal{R}$;
(3) if p is a sending state then

[1] In definitions for syntactical session types, e.g., [MY15], input contexts are used to accumulate inputs that precede anticipated outputs; here, having no specific syntax for inputs, we use input trees instead.

(a) if $T = q$ and q is a sending state, then
$\forall p' \in Q_1 \; s.t. \; p \xrightarrow{!a} p'. \; \exists q' \in Q_2 \; s.t. \; q \xrightarrow{!a} q' \wedge (p', q') \in \mathcal{R}$;

(b) otherwise, if $T = \mathcal{A}[q_i]^{i \in I}$ then $\neg \mathsf{cycle}(!, p)$ and $\forall i \in I. \mathsf{inTree}(q_i) = \mathcal{A}_i[q_{i,h}]^{h \in H_i}$ and $\forall p' \in Q_1 \; s.t. \; p \xrightarrow{!a} p'$.

$$\forall i \in I. \forall h \in H_i. \; \exists q'_{i,h} \in Q_2 \; s.t. \; q_{i,h} \xrightarrow{!a} q'_{i,h} \wedge (p', \mathcal{A}[\mathcal{A}_i[q'_{i,h}]^{h \in H_i}]^{i \in I}) \in \mathcal{R}.$$

M_1 is an asynchronous subtype of M_2, written $M_1 \preccurlyeq M_2$, if there is an asynchronous subtyping relation $\mathcal{R}$ such that $(q_{0_1}, q_{0_2}) \in \mathcal{R}$.

The relation $M_1 \preccurlyeq M_2$ checks that M_1 is a subtype of M_2 by executing M_1 and simulating its execution with M_2. M_1 may fire send actions earlier than M_2, in which case M_2 is allowed to fire these actions even if it needs to fire some receive actions first. These receive actions are accumulated in an input context and are expected to be subsequently matched by M_1. Due to the presence of such an input context, the states reached by M_2 during the computation are represented as input trees. The definition first differentiates the type of state p:

Final: Case (1) says that if M_1 is in a final state, then M_2 is in a final state with an empty input context.

Receiving: Case (2) says that if M_1 is in a receiving state, then either (2a) the input context is empty ($T = q$) and M_1 must be able to receive all messages that M_2 can receive; or, (2b) M_1 must be able to consume all the messages at the root of the input tree.

Sending: Case (3) applies when M_1 is in a sending state, there are two sub-cases.
Case (3a) says that if the input context is empty ($T = q$) and q is also a sending state, then M_2 must be able to send all messages that M_1 can send. If this sub-case above does not apply (i.e., the input context is not empty or q is not a sending state), then the one below must hold.
Case (3b) enforces correct output anticipation, i.e., M_2 must be able to send every a that M_1 can send after some receive actions recorded in each $\mathcal{A}_i[q_{i,h}]^{h \in H_i}$. Note that whichever receiving path M_2 chooses, it must be able to send all possible output actions $!a$ of M_1, i.e., $!a$ should be available at the end of each receiving path. Moreover, given that there are accumulated inputs, we require that $\mathsf{cycle}(!, p)$ does *not* hold, guaranteeing that subtyping preserves orphan-message freedom, i.e., such accumulated receive actions will be eventually executed.

Observe that Case (2) enforces a form of contra-variance for receive actions, while Case (3) enforces a form of covariance for send actions.

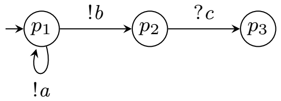
Example 2.7. Consider M_C and M_R from Figures 1 and 2, we have $M_R \preccurlyeq M_C$ (see § 3). A fragment of the relation $\mathcal{R}$ from Definition 2.6 is given in Figure 4. Considering the identifier (bottom left) of each node in Figure 4, we have:

- Case (1) of Definition 2.6 does not apply to any configuration in this example (there is no final node in these machines).
- Case (2a) applies to node n_1, i.e., $q_2 \preccurlyeq q_2$ (note that q_2 are receiving states in both machines).
- Case (2b) applies to nodes n_5, n_9, and n_{13}; where q_2 of machine M_R is a receiving state and the input context is not empty.
- Case (3a) applies to nodes n_0, n_2, and n_3, where the input context is empty and both states are sending states.

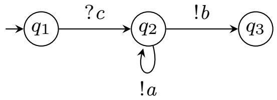
- Case (3b) applies to nodes n_4, n_6, n_7, n_8, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, and n_{16}. Observe that this case does not require the input context to be non-empty (e.g., n_4), and that the condition $\neg\mathsf{cycle}(!, p)$ holds for all states p in M_R since there is no send-only cycle in this machine.

Example 2.8. For the two machines below, we have $M_1 \not\preccurlyeq M_2$ and $M_2 \not\preccurlyeq M_1$:

M_1:



M_2:



For the $M_1 \not\preccurlyeq M_2$ case consider the initial configuration (p_1, q_1). Since p_1 is a sending state, but q_1 is a receiving state, Case (3b) *appears* to be the only applicable case of Definition 2.6. However, we have $\mathsf{cycle}(!, p_1)$ hence $(p_1, q_1) \notin \mathcal{R}$, for every asynchronous subtyping relation $\mathcal{R}$.

For the $M_2 \not\preccurlyeq M_1$ case, consider the initial configuration (q_1, p_1). Since q_1 is a receiving state, only Case 2 would be applicable. However, the input context is empty and p_1 is a sending state, therefore neither Case (2a) nor Case (2b) apply hence $(q_1, p_1) \notin \mathcal{R}$, for every asynchronous subtyping relation $\mathcal{R}$.

3. A Sound Algorithm for Asynchronous Subtyping

Our subtyping algorithm takes two machines M_1 and M_2 then produces three possible outputs: *true*, *false*, or *unknown*, which respectively indicate that $M_1 \preccurlyeq M_2$, $M_1 \not\preccurlyeq M_2$, or that the algorithm was unable to prove either of these two results. The algorithm consists of three stages. (1) It builds the *simulation tree* of M_1 and M_2 (see Definition 3.2) that represents sequences of checks between M_1 and M_2, corresponding to the checks in the definition of asynchronous subtyping. Simulation trees may be infinite, but the construction terminates whenever: either it reaches a node that cannot be expanded, it visits a node whose label has been seen along the path from the root, or it expands a node whose ancestors validate a termination condition that we formalise in Theorem 3.8. The resulting tree satisfies one of the following conditions: (i) it contains a leaf that could not be expanded because the node represents an unsuccessful check between M_1 and M_2 (in which case the algorithm returns *false*), (ii) all leaves are successful final configurations, see Condition (1) of Definition 2.6, in which case the algorithm replies *true*, or (iii) for each leaf n it is possible to identify a corresponding ancestor $\mathsf{anc}(n)$. In this last case the tree and the identified ancestors are passed onto the next stage. (2) The algorithm divides the finite tree into several subtrees rooted at those ancestors that do not have other ancestors above them (see the strategy that we outline on page 16). (3) The final stage analyses whether each subtree is of one of the two following kinds. (i) All the leaves in the subtree have the same label as their ancestors: in this case all checks required to verify subtyping have been performed. (ii) The subtree is a *witness subtree* (see Definition 3.16), meaning that all the checks that may be considered in any extension of the finite subtree are guaranteed to be successful as well. If all the identified subtrees are of one of these two kinds, the algorithm replies *true*. Otherwise, it replies *unknown*.

3.1. **Generating Asynchronous Simulation Trees.** We first define labelled trees, of which our simulation trees are instances; then, we give the operational rules for generating a simulation tree from a pair of communicating machines.

Definition 3.1 (Labelled Tree)**.** A labelled tree is a tree[2] $(N, n_0, \hookrightarrow, \mathcal{L}, \Sigma, \Gamma)$, consisting of nodes N, root $n_0 \in N$, edges $\hookrightarrow \subseteq N \times \Sigma \times N$, and node labelling function $\mathcal{L} : N \longmapsto \Gamma$.

Hereafter, we write $n \xhookrightarrow{\sigma} n'$ when $(n, \sigma, n') \in \hookrightarrow$ and write $n_1 \xhookrightarrow{\sigma_1 \cdots \sigma_k} n_{k+1}$ when there are $n_1, \ldots, n_{k+1}$, such that $n_i \xhookrightarrow{\sigma_i} n_{i+1}$ for all $1 \leq i \leq k$. We write $n \hookrightarrow n'$ when $n \xhookrightarrow{\sigma} n'$ for some σ and the label is not relevant. As usual, we write $\hookrightarrow^*$ for the reflexive and transitive closure of $\hookrightarrow$, and $\hookrightarrow^+$ for its transitive closure. Moreover, we reason up-to tree isomorphism, i.e., two labelled trees are equivalent if there exists a bijective node renaming that preserves both node labelling and labelled transitions.

We can then define simulation trees, labelled trees representing all possible configurations reachable by the simulation checked by asynchronous session subtyping.

Definition 3.2 (Simulation Tree)**.** Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines. The simulation tree of M_1 and M_2, written $\mathsf{simtree}(M_1, M_2)$, is a labelled tree $(N, n_0, \hookrightarrow, \mathcal{L}, \mathit{Act}, P \times \mathcal{T}_Q)$. The labels $(p, T) \in (P \times \mathcal{T}_Q)$ are denoted also with $p \preccurlyeq T$. In order to define $\hookrightarrow$ and $\mathcal{L}$, we first consider an Act-labelled relation on $(P \times \mathcal{T}_Q)$, with elements denoted with $p \preccurlyeq T \xhookrightarrow{\ell} p' \preccurlyeq T'$, defined as the minimal relation satisfying the following rules:

$$\frac{p \xrightarrow{?a} p' \quad q \xrightarrow{?a} q' \quad \mathsf{in}(p) \supseteq \mathsf{in}(q)}{p \preccurlyeq q \xhookrightarrow{?a} p' \preccurlyeq q'} \ (\mathsf{In}) \qquad \frac{p \xrightarrow{!a} p' \quad q \xrightarrow{!a} q' \quad \mathsf{out}(p) \subseteq \mathsf{out}(q)}{p \preccurlyeq q \xhookrightarrow{!a} p' \preccurlyeq q'} \ (\mathsf{Out})$$

$$\frac{p \xrightarrow{?a_k} p' \quad k \in I \quad \mathsf{in}(p) \supseteq \{a_i \mid i \in I\}}{p \preccurlyeq \langle a_i : T_i \rangle_{i \in I} \xhookrightarrow{?a_k} p' \preccurlyeq T_k} \ (\mathsf{InCtx})$$

$$\frac{\begin{array}{c} p \xrightarrow{!a} p' \qquad \neg\mathsf{cycle}(!, p) \\ \forall j \in J. \big(\mathsf{inTree}(q_j) = \mathcal{A}_j[q_{j,h}]^{h \in H_j} \wedge \forall h \in H_j. (\mathsf{out}(p) \subseteq \mathsf{out}(q_{j,h}) \wedge q_{j,h} \xrightarrow{!a} q'_{j,h})\big) \end{array}}{p \preccurlyeq \mathcal{A}[q_j]^{j \in J} \xhookrightarrow{!a} p' \preccurlyeq \mathcal{A}[\mathcal{A}_j[q'_{j,h}]^{h \in H_j}]^{j \in J}} \ (\mathsf{OutAcc})$$

We now define $\hookrightarrow$ and $\mathcal{L}$ as the transition relation and the labelling function s.t. $\mathcal{L}(n_0) = p_0 \preccurlyeq q_0$ and, for each $n \in N$ with $\mathcal{L}(n) = p \preccurlyeq T$, the following holds:

- if $p \preccurlyeq T \xhookrightarrow{\ell} p' \preccurlyeq T'$ then there exists a unique n' s.t. $n \xhookrightarrow{\ell} n'$ with $\mathcal{L}(n') = p' \preccurlyeq T'$;
- if $n \xhookrightarrow{\ell} n'$ with $\mathcal{L}(n') = p' \preccurlyeq T'$ then $p \preccurlyeq T \xhookrightarrow{\ell} p' \preccurlyeq T'$.

Notice that such a tree exists (it can be constructed inductively starting from the root n_0) and it is unique (up-to tree isomorphism).

Given machines M_1 and M_2, Definition 3.2 generates a tree whose nodes are labelled by terms of the form $p \preccurlyeq \mathcal{A}[q_i]^{i \in I}$ where p represents the state of M_1, $\mathcal{A}$ represents the receive actions accumulated by M_2, and each q_i represents the state of machine M_2 after each path of accumulated receive actions from the root of $\mathcal{A}$ to the i^{th} hole. Note that we overload the

[2] A tree is a connected directed graph without cycles: $\forall n \in N.\ n_0 \hookrightarrow^* n \wedge \forall n, n' \in N.\ n \hookrightarrow^+ n'.\ n \neq n'$.

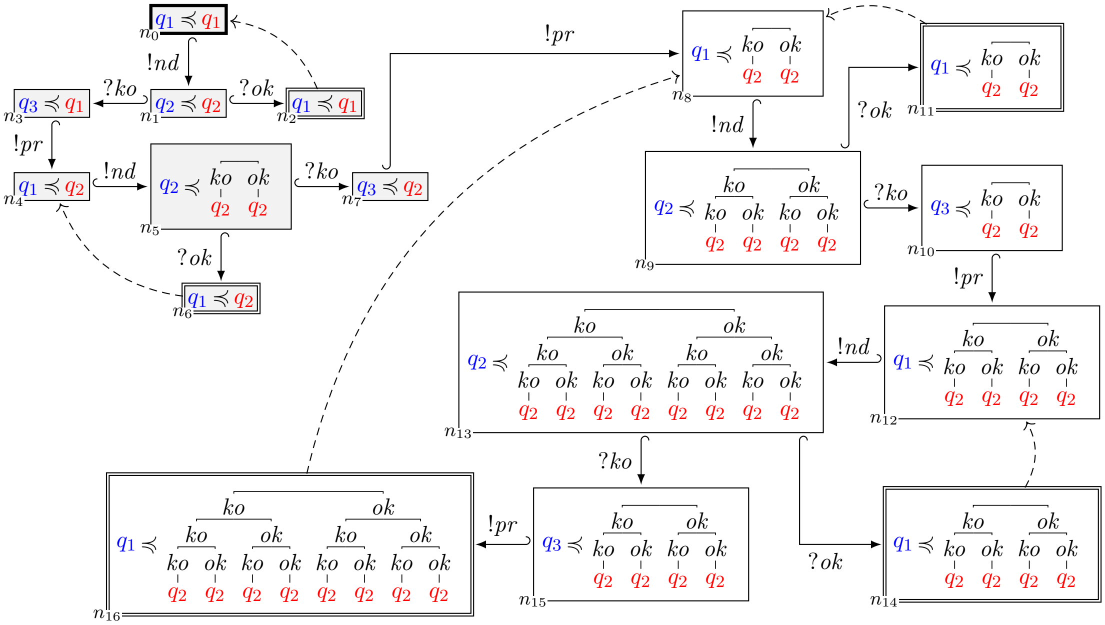

n_0 $q_1 \preccurlyeq q_1$
!nd
n_1 $q_2 \preccurlyeq q_2$
?ko
n_3 $q_3 \preccurlyeq q_1$
?ok
n_2 $q_1 \preccurlyeq q_1$
!pr
n_4 $q_1 \preccurlyeq q_2$
!nd
n_5
?ko
n_7 $q_3 \preccurlyeq q_2$
?ok
n_6 $q_1 \preccurlyeq q_2$
!pr
n_8
!nd
n_9
?ok
n_{11}
?ko
n_{10}
!pr
n_{12}
!nd
n_{13}
?ko
n_{15}
!pr
n_{16}
?ok
n_{14}
ko
ok
q_2

Figure 4: Part of the simulation tree (solid edges only) and candidate tree for $M_R \preccurlyeq M_C$ (Figure 1 and 2). The root is circled in thicker line. The node identities are shown at the bottom left of each label.

symbol $\preccurlyeq$ used for asynchronous subtyping (Definition 2.6), however the actual meaning is always made clear by the context. We comment each rule in detail below.

Rules (In) and (Out) enforce contra-variance of inputs and covariance of outputs, respectively, when no accumulated receive actions are recorded, i.e., $\mathcal{A}$ is a single hole. Rule (In) corresponds to Case (2a) of Definition 2.6, while rule (Out) corresponds to Case (3a).

Rule (InCtx) is applicable when the input tree $\mathcal{A}$ is non-empty and the state p (of M_1) is able to perform a receive action corresponding to any message located at the root of the input tree (contra-variance of receive actions). This rule corresponds to Case (2b) of Definition 2.6.

Rule (OutAcc) allows M_2 to execute some receive actions before matching a send action executed by M_1. This rule corresponds to Case (3b) of Definition 2.6. Intuitively, each send action outgoing from state p must also be eventually executable from each of the states q_j (in M_2) which occur in the input tree $\mathcal{A}[q_j]^{j \in J}$. The possible combinations of receive actions executable from each q_j before executing $!a$ is recorded in $\mathcal{A}_j$, using $\mathsf{inTree}(q_j)$. We assume that the premises of this rule only hold when all invocations of $\mathsf{inTree}(\cdot)$ are defined. Each tree of accumulated receive actions is appended to its respective branch of the input context $\mathcal{A}$, using the notation $\mathcal{A}[\mathcal{A}_j[q'_{j,h}]^{h \in H_j}]^{j \in J}$. The premise $\mathsf{out}(p) \subseteq \mathsf{out}(q_{j,h}) \wedge q_{j,h} \xrightarrow{!a} q'_{j,h}$ guarantees that each $q_{j,h}$ can perform the send actions available from p (covariance of send actions). The additional premise $\neg\mathsf{cycle}(!, p)$ corresponds to that of Case (3b) of Definition 2.6.

Example 3.3. Figure 4 gives a graphical view of the initial part of the simulation tree $\mathsf{simtree}(M_R, M_C)$. Consider the solid edges only for now, they correspond to the $\hookrightarrow$-relation. Observe that all branches of the simulation tree are infinite; some traverse nodes with infinitely many different labels, due to the unbounded growth of the input trees (e.g., the one repeatedly performing transitions $!nd{\cdot}?ko{\cdot}!pr$); while others traverse nodes with *finitely* many distinct labels (e.g., the one performing first transitions $!nd{\cdot}?ko{\cdot}!pr$ and then repeatedly performing $!nd{\cdot}?ok$).

We adapt the terminology of [JM99] and say that a node n of $\mathsf{simtree}(M_1, M_2)$ is a *leaf* if it has no successors. A leaf n is *successful* iff $\mathcal{L}(n) = p \preccurlyeq q$, with p and q final; all other leaves are unsuccessful. A *branch* (a full path through the tree) is *successful* iff it is infinite or finishes with a successful leaf; otherwise it is unsuccessful. Using this terminology, we relate asynchronous subtyping (Definition 2.6) with simulation trees (Definition 3.2) in Theorem 3.4.

Theorem 3.4. *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines. All branches in $\mathsf{simtree}(M_1, M_2)$ are successful if and only if $M_1 \preccurlyeq M_2$.*

Proof. We start from the *if part.* Consider two communicating machines $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ such that $M_1 \preccurlyeq M_2$. By definition of $M_1 \preccurlyeq M_2$, we have that there exists an asynchronous subtyping $\mathcal{R}$ such that $(p_0, q_0) \in \mathcal{R}$. Consider now $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$, having a root labelled with $p_0 \preccurlyeq q_0$. We have that also other nodes $n \in N$ are such that $\mathcal{L}(n) = p \preccurlyeq T$ implies $(p, T) \in \mathcal{R}$. This is easily proved by induction on the length of the sequence of transitions $n_0 \hookrightarrow^+ n$, observing that the rules for the construction of the simulation tree check on p and T the same properties checked by the definition of asynchronous session subtyping, and generate new transitions to nodes labelled with $p' \preccurlyeq T'$ corresponding to the pairs (p', T') that are required to be in $\mathcal{R}$. This guarantees that, for every n in the simulation tree, either $\mathcal{L}(n) = p \preccurlyeq q$ with $p \not\rightarrow$ and $q \not\rightarrow$ (i.e., p and q are final) implying that the branch to n is successful, or there exists n' such that $n \hookrightarrow n'$. This guarantees that in $\mathsf{simtree}(M_1, M_2)$ there exists no unsuccessful branch.

We now move to the *only if part.* Consider two communicating machines $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ and their simulation tree $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$. Consider now the relation $\mathcal{R} \subseteq P \times \mathcal{T}_Q$ such that $(p, T) \in \mathcal{R}$ if and only if there exists $n \in N$ s.t. $\mathcal{L}(n) = p \preccurlyeq T$. With similar arguments as in the above case, we prove that $\mathcal{R}$ is an asynchronous subtyping relation. Hence, given that $\mathcal{L}(n_0) = p_0 \preccurlyeq q_0$, we have $(p_0, q_0) \in \mathcal{R}$, hence also $M_1 \preccurlyeq M_2$. □

3.2. **A Simulation Tree-Based Algorithm.** A consequence of the undecidability of asynchronous session subtyping [LY17, BCZ18, BCZ17] is that checking whether all branches in $\mathsf{simtree}(M_1, M_2)$ are successful is undecidable. The problem follows from the presence of infinite branches that cannot be algorithmically identified. Our approach is to characterise finite subtrees (called *witness subtrees*) such that all the branches that traverse these finite subtrees are guaranteed to be infinite.

The presentation of our algorithm is in three parts. In Part (1), we give the definition of the kind of *finite* subtree (of a simulation tree) we are interested in (called *candidate* subtrees). In Part (2), we give an algorithm to extract *candidate* subtrees from a simulation tree $\mathsf{simtree}(M_1, M_2)$. In Part (3) we show how to check whether a candidate subtree (which is finite) is a *witness* of infinite branches (hence successful) in the simulation tree.

3.2.1. *Part 1. Characterising finite and candidate sub-trees.* We define the candidate subtrees of a simulation tree, which are finite subtrees accompanied by an ancestor function mapping each boundary node n to a node located on the path from the root of the tree to n.

Definition 3.5 (Finite Subtree)**.** A finite subtree (r, B) of a labelled tree $S = (N, n_0, \hookrightarrow, \mathcal{L}, \Sigma, \Gamma)$, with r being the subtree root and $B \subseteq N$ the finite set of its leaves (boundary nodes), is the subgraph of S such that:

(1) $\forall n \in B.\ r \hookrightarrow^* n$;
(2) $\forall n \in B.\ \nexists n' \in B.\ n \hookrightarrow^+ n'$; and
(3) $\forall n \in N.\ r \hookrightarrow^* n \implies \exists n' \in B.\ n \hookrightarrow^* n' \vee n' \hookrightarrow^* n$.

We use $\mathsf{nodes}(S, r, B) = \{n \in N \mid \exists n' \in B.\ r \hookrightarrow^* n \hookrightarrow^* n'\}$ to denote the (finite) set of nodes of the finite subtree (r, B). Notice that $r \in \mathsf{nodes}(S, r, B)$ and $B \subseteq \mathsf{nodes}(S, r, B)$.

Condition (1) requires that each boundary node can be reached from the root of the subtree. Condition (2) guarantees that the boundary nodes are not connected, i.e., they are on different paths from the root. Condition (3) enforces that each branch of the tree passing through the root r contains a boundary node.

Definition 3.6 (Candidate Subtree)**.** Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$.
A *candidate subtree* of $\mathsf{simtree}(M_1, M_2)$ is a finite subtree (r, B) paired with a function $\mathsf{anc} : B \longmapsto \mathsf{nodes}(\mathsf{simtree}(M_1, M_2), r, B) \backslash B$ such that, for all $n \in B$, we have: $\mathsf{anc}(n) \hookrightarrow^+ n$ and there are $p, \mathcal{A}, \mathcal{A}', I, J, \{q_j \mid j \in J\}$ and $\{q_i \mid i \in I\}$ such that

$$\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i \in I} \quad \wedge \quad \mathcal{L}(\mathsf{anc}(n)) = p \preccurlyeq \mathcal{A}'[q_j]^{j \in J} \quad \wedge \quad \{q_i \mid i \in I\} \subseteq \{q_j \mid j \in J\}$$

A candidate subtree is a finite subtree accompanied by a total function on its boundary nodes. The purpose of function anc is to map each boundary node n to a "similar" ancestor n' such that: n' is a node (different from n) on the path from the root r to n (recall that we have $r \notin B$) such that the labels of n' and n share the same state p of M_1, and the states of M_2 (that populate the holes in the leaves of the input context of the boundary node) are a subset of those considered for the ancestor. Given a candidate subtree, we write $img(\mathsf{anc})$ for the set $\{n \mid \exists n' \in B.\ \mathsf{anc}(n') = n\}$, i.e., $img(\mathsf{anc})$ is the set of ancestors in the candidate subtree.

Example 3.7. Figure 4 depicts a finite subtree of $\mathsf{simtree}(M_R, M_C)$. We can distinguish several distinct candidate subtrees in Figure 4. For instance one subtree is rooted at n_0, and its boundary nodes are $\{n_2, n_6, n_{11}, n_{14}, n_{16}\}$; another subtree is rooted at n_8 and its boundary nodes are $\{n_{11}, n_{14}, n_{16}\}$ (boundary nodes are highlighted with a double border). In each subtree, the anc function is represented by the dashed edges from its boundary nodes to their respective ancestors.

3.2.2. *Part 2. Identifying candidate subtrees.* We now describe how to generate a finite subtree of the simulation tree, from which we extract candidate subtrees. Since simulation trees are potentially infinite, we need to identify termination conditions (i.e., conditions on nodes that become the boundary of the generated finite subtree).

We first need to define the auxiliary function $\mathsf{extract}(\mathcal{A}, \omega)$, which checks the presence of a sequence of messages ω in an input context $\mathcal{A}$, and extracts the residual input context.

$$\mathsf{extract}(\mathcal{A}, \omega) = \begin{cases} \mathcal{A} & \text{if } \omega = \epsilon \\ \mathsf{extract}(\mathcal{A}_i, \omega') & \text{if } \omega = a_i \cdot \omega', \mathcal{A} = \langle a_j : \mathcal{A}_j \rangle_{j \in J}, \text{ and } i \in J \\ \bot & \text{otherwise} \end{cases}$$

Our termination condition is formalised in Theorem 3.8 below. This result follows from an argument based on the finiteness of the states of M_1 and of the sets of states from M_2 (which populate the holes of the input contexts in the labels of the nodes in the simulation tree). We write $\mathsf{minHeight}(\mathcal{A})$ for the smallest $\mathsf{height}_i(\mathcal{A})$, with $i \in I(\mathcal{A})$, where $\mathsf{height}_i(\mathcal{A})$ is the length of the path from the root of the input context $\mathcal{A}$ to the i^{th} hole.

Theorem 3.8. *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$.*
For each infinite path $n_0 \hookrightarrow n_1 \hookrightarrow n_2 \cdots \hookrightarrow n_l \hookrightarrow \cdots$ there exist $i < j < k$, with

$$\mathcal{L}(n_i) = p \preccurlyeq \mathcal{A}_i[q_h]^{h \in H_i} \qquad \mathcal{L}(n_j) = p \preccurlyeq \mathcal{A}_j[q'_h]^{h \in H_j} \qquad \mathcal{L}(n_k) = p \preccurlyeq \mathcal{A}_k[q''_h]^{h \in H_k}$$

such that $\{q'_h \mid h \in H_j\} \subseteq \{q_h \mid h \in H_i\}$ and $\{q''_h \mid h \in H_k\} \subseteq \{q_h \mid h \in H_i\}$;
and, for $n_i \overset{\psi}{\hookrightarrow} n_j$:

(*i*) $\mathsf{rcv}(\psi) = \omega_1 \cdot \omega_2$ *with ω_1 s.t.* $\exists t, z.\ \mathsf{extract}(\mathcal{A}_i, \omega_1) = [\,]_t \wedge \mathsf{extract}(\mathcal{A}_k, \omega_1) = [\,]_z$, *or*
(*ii*) $\mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_i, \mathsf{rcv}(\psi))) \leq \mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_k, \mathsf{rcv}(\psi)))$.

Proof. Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$, and let $n_0 \hookrightarrow n_1 \hookrightarrow n_2 \cdots \hookrightarrow n_l \hookrightarrow \cdots$ be an infinite path in the simulation tree. For each n_i, let $\mathcal{S}_i$ be the pair (p_i, R_i), with $p_i \in P$ and $R_i \subseteq Q$, such that $\mathcal{L}(n_i) = p_i \preccurlyeq \mathcal{A}_i[q_j]^{j \in J_i}$ and $R_i = \{q_j \mid j \in J_i\}$. Notice that there are at most $|P| \times 2^{|Q|}$ distinct pairs (p_i, R_i), in which p_i is an element taken from the finite set P, and R_i is a subset of the finite set Q. This guarantees the existence of infinite pairs of nodes $(n_{i_1}, n_{i'_1}), (n_{i_2}, n_{i'_2}), \ldots, (n_{i_j}, n_{i'_j}), \ldots$ taken from the above infinite path, such that, for all j:

- $\mathcal{S}_{i_j} = \mathcal{S}_{i'_j}$ and
- $i_j < i'_j < i_{j+1}$ and
- $i'_j - i_j \leq |P| \times 2^{|Q|}$.

The above follows from the possibility to repeatedly select, by following from left to right the infinite sequence $n_0 \hookrightarrow n_1 \hookrightarrow n_2 \cdots \hookrightarrow n_l \hookrightarrow \cdots$, the first occurring pair (n_k, n_l), with $k < l$, such that $\mathcal{S}_k = \mathcal{S}_l$. Being the first pair of this type that occurs, we have that $l - k \leq |P| \times 2^{|Q|}$.

For the above infinite list of pairs $(n_{i_1}, n_{i'_1}), (n_{i_2}, n_{i'_2}), \ldots, (n_{i_j}, n_{i'_j}), \ldots$ let ψ_j be such that $n_{i_j} \overset{\psi_j}{\hookrightarrow} n_{i'_j}$. All these infinitely many sequences of actions ψ_j have bounded length (smaller than $|P| \times 2^{|Q|}$), hence infinitely many of them will coincide (this is because there are only boundedly many distinct actions that are admitted). Let α be such a sequence of actions that is considered for infinitely many paths $n_{i_j} \overset{\alpha}{\hookrightarrow} n_{i'_j}$. Moreover, being the possible distinct (p_i, Q_i) finite, there exists one pair (p, Q) such that infinitely many of these paths $n_{i_j} \overset{\alpha}{\hookrightarrow} n_{i'_j}$ will be such that $\mathcal{S}_{i_j} = \mathcal{S}_{i'_j} = (p, Q)$.

Summarising, we have proved the existence of $(n_{v_1}, n_{v'_1}), (n_{v_2}, n_{v'_2}), \dots, (n_{v_j}, n_{v'_j}), \dots$, with $v_j < v'_j < v_{j+1}$ for all j, for which there exist (p, Q) and α such that, for all j, $n_{v_j} \overset{\alpha}{\hookrightarrow} n_{v'_j}$ and $\mathcal{S}_{v_j} = \mathcal{S}_{v'_j} = (p, Q)$.

We now consider $\omega = \mathsf{rcv}(\alpha)$. We have that the input actions in ω, executed in each path $n_{v_j} \overset{\alpha}{\hookrightarrow} n_{v'_j}$, will be matched by the input context $\mathcal{A}_j$ of $\mathcal{L}(n_{v_j}) = p \preccurlyeq \mathcal{A}_j[q_j]^{h\in H}$. There are two possibilities:

(1) either ω is included in a path root-hole of $\mathcal{A}_j$ (hence $\mathsf{extract}(\mathcal{A}_j, \omega)$ is defined),
(2) or there exists a path root-hole which corresponds to a prefix of ω (in this case we have that $\omega = \omega_1 \cdot \omega_2$ with $\mathsf{extract}(\mathcal{A}_j, \omega_1) = [\,]_{t_j}$).

At least one of the two cases occurs infinitely often, i.e., there exist infinitely many indices j, such that for all paths $n_{v_j} \overset{\alpha}{\hookrightarrow} n_{v'_j}$ item 1 holds, or there exist infinitely many indices j, such that for all paths $n_{v_j} \overset{\alpha}{\hookrightarrow} n_{v'_j}$ item 2 holds. In the first case, we have that there exist at least two indices j_1 and j_2 such that $\mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_{j_1}, \omega)) \leq \mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_{j_2}, \omega))$ (in fact, $\mathsf{minHeight}()$ returns a non negative value, hence such values cannot infinitely decrease). In the second case, we have that there exist at least two indices j_1 and j_2 such that $\mathsf{extract}(\mathcal{A}_{j_1}, \omega_1) = [\,]_{t_{j_1}}$ and $\mathsf{extract}(\mathcal{A}_{j_2}, \omega_1) = [\,]_{t_{j_2}}$ for the same ω_1 prefix of ω (in fact, ω has only finitely many prefixes).

We can conclude that the thesis holds by considering $i = v_{j_1}$, $j = v'_{j_1}$ and $k = v_{j_2}$. □

Intuitively, the theorem above says that for each infinite branch in the simulation tree, we can find special nodes n_i, n_j and n_k such that the set of states in $\mathcal{A}_j$ (resp. $\mathcal{A}_k$) is included in that of $\mathcal{A}_i$ and the receive actions in the path from n_i to n_j are such that: either (i) only a precise prefix of such actions will be taken from the receive actions accumulated in n_i and n_k or (ii) all of them will be taken from the receive actions in which case n_k must have accumulated more receive actions than n_i. Case (i) deals with infinite branches with only finite labels (hence finite accumulation) while case (ii) considers those cases in which there is unbounded accumulation along the infinite branch.

As an example of this latter case, consider the simulation tree depicted in Figure 4. Let $n_i = n_8$, $n_j = n_{12}$ and $n_k = n_{16}$. These nodes are along the same path, moreover we have $\mathcal{L}(n_i) = q_1 \preccurlyeq \mathcal{A}_i[q_h]^{h\in H_i}$, $\mathcal{L}(n_j) = q_1 \preccurlyeq \mathcal{A}_j[q'_h]^{h\in H_j}$, $\mathcal{L}(n_k) = q_1 \preccurlyeq \mathcal{A}_k[q''_h]^{h\in H_k}$ with $\{q_h \mid h \in H_i\} = \{q'_h \mid h \in H_j\} = \{q''_h \mid h \in H_k\} = \{q_2\}$ and $0 = \mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_i, ?ko)) \leq \mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_k, ?ko)) = 2$. Notice that the path in the simulation tree from n_8 to n_{16} can be infinitely repeated with the effect of increasing the height of the input context.

Based on Theorem 3.8, the following *algorithm* generates a finite subtree of $\mathsf{simtree}(M_1, M_2)$:

> Starting from the root, compute the branches[3] of $\mathsf{simtree}(M_1, M_2)$ stopping when one of the following types of node is encountered: a leaf, a node n with a label already seen along the path from the root to n, or a node n_k (with the corresponding node n_i) as those described by the above Theorem 3.8.

Example 3.9. Consider the finite subtree in Figure 4. It is precisely the finite subtree identified as described above: we stop generating the simulation tree at nodes n_2, n_6, n_{11}, and n_{14} (because their labels have been already seen at the corresponding ancestors n_0, n_4, n_8, and n_{12}) and n_{16} (because of the ancestors n_8 and n_{12} such that n_8, n_{12} and n_{16} correspond to the nodes n_i, n_j and n_k of Theorem 3.8).

[3] The order nodes are generated is not important (our implementation uses a DFS approach, cf. §4).

When the computed finite subtree contains an unsuccessful leaf, we can immediately conclude that the considered communicating machines are not related. Otherwise, we extract smaller finite subtrees (from the subtree) that are potential candidates to be subsequently checked.

> We define the anc function as follows: for boundary nodes n with an ancestor n' such that $\mathcal{L}(n) = \mathcal{L}(n')$ we define $\mathsf{anc}(n) = n'$; for boundary nodes n_k with the corresponding node n_i as those described by Theorem 3.8, we define $\mathsf{anc}(n_k) = n_i$. The extraction of the finite subtrees is done by characterising their roots (and taking as boundary their reachable boundary nodes): let $P = \{n \in img(\mathsf{anc}) \mid \exists n'.\ \mathsf{anc}(n') = n \wedge \mathcal{L}(n) \neq \mathcal{L}(n')\}$, the set of such roots is $R = \{n \in P \mid \nexists n' \in P.\ n' \hookrightarrow^{+} n\}$.

Intuitively, to extract subtrees, we restrict our attention to the set P of ancestors with a label different from their corresponding boundary node (corresponding to branches that can generate unbounded accumulation). We then consider the forest of subtrees rooted in nodes in P without an ancestor in P. Notice that for successful leaves we do not define anc; hence, only extracted subtrees without successful nodes have a completely defined anc function. These are candidate subtrees that will be checked as described in the next step.

Example 3.10. Consider the finite subtree in Figure 4. Following the strategy above we extract from it the candidate subtree rooted at n_8 (white nodes), with boundary $\{n_{11}, n_{14}, n_{16}\}$. Note that each ancestor node above n_8 has a label identical to its boundary node.

3.2.3. *Part 3. Checking whether the candidate subtrees are witnesses of infinite branches.*

The final step of our algorithm consists in verifying a property on the identified candidate subtrees which guarantees that all branches traversing the root of the candidate subtree are infinite, hence successful. A candidate subtree satisfies this property when it is also a *witness subtree*, which is the key notion (Definition 3.16) presented in this third part.

In order for a subtree to be a witness, we require that any behaviour in the simulation tree going beyond the subtree is the infinite repetition of the behaviour already observed in the considered finite subtree. This infinite repetition is only possible if whatever receive actions are accumulated in the input context $\mathcal{A}$ (using Rule (OutAcc)) are eventually executed by the candidate subtype M_1 in Rule (InCtx). The compatibility check between the receive actions that can be accumulated and the receive actions that are eventually executed is done by first synthesising a finite representation of the possible (repeated) accumulation of the candidate supertype M_2 and the possible (repeated) receive actions of the candidate subtype M_1. We then check whether these representations of the input actions are *compatible*, wrt. the $\sqsubseteq$-relation, see Definition 3.12. We define these representations of the input behaviours as a system of (possibly) mutually recursive equations, which we call a *system of input tree equations.*

Intuitively, a system of input tree equations represents a family of trees, that we use to represent the input behaviour of types. We need to consider families of trees because types include also output actions that, in case we are concerned with input actions only, can be seen as internal silent actions, representing nondeterministic choices among alternative future inputs (i.e. alternative subtrees).

Definition 3.11 (Input Tree Equations)**.** Given a set of variables $\mathcal{V}$, ranged over by X, an input tree expression is a term of the grammar

$$E \quad ::= \quad X \ \mid \ \langle a_i : E_i \rangle_{i \in I} \ \mid \ \langle E_i \rangle_{i \in I}$$

The free variables of an input tree expression E are the variables which occur in E. Let $T_{\mathcal{V}}$ be the set of input tree expressions whose free variables are in $\mathcal{V}$.

A system of input tree equations is a tuple $\mathcal{G} = (\mathcal{V}, X_0, \mathbf{E})$ consisting of a set of variables $\mathcal{V}$, an initial variable $X_0 \in \mathcal{V}$, and with $\mathbf{E}$ consisting of exactly one input tree expression $X \overset{\text{def}}{=} E$ for each $X \in \mathcal{V}$, with $E \in \mathcal{T}_{\mathcal{V}}$.

Given an input tree expression of the form $\langle a_i : E_i \rangle_{i \in I}$ or $\langle E_i \rangle_{i \in I}$, we assume that $I \neq \emptyset$, $\forall i \neq j \in I.\ a_i \neq a_j$, and that the order of the sub-terms is irrelevant. Whenever convenient, we use set-builder notation to construct an input tree expression, e.g., $\langle E_i \mid i \in I \rangle$. In an input tree equation, the construct $\langle a_i : E_i \rangle_{i \in I}$ represents the capability of accumulating (or actually executing) the receive actions on each message a_i then behaving as in E_i. The construct $\langle E_i \rangle_{i \in I}$ represents a *silent choice* between the different capabilities E_i.

We now define the notion of compatibility between two systems of input tree equations. Intuitively, two systems of input tree equations are compatible when all the trees of the former have *less* alternatives than the trees of the latter. More precisely, at each input choice, the alternative branchings of the former are included in those of the latter.

Definition 3.12 (Input Tree Compatibility)**.** Given two systems of input tree equations $\mathcal{G} = (\mathcal{V}, X_0, \mathbf{E})$ and $\mathcal{G}' = (\mathcal{V}', X_0', \mathbf{E}')$, such that $\mathcal{V} \cap \mathcal{V}' = \emptyset$, we say that $\mathcal{G}$ is *compatible* with $\mathcal{G}'$, written $\mathcal{G} \sqsubseteq \mathcal{G}'$, if there exists a compatibility relation $\mathcal{R} \subseteq \mathcal{T}_{\mathcal{V}} \times \mathcal{T}'_{\mathcal{V}}$. That is a relation $\mathcal{R}$ s.t. $(X_0, X_0') \in \mathcal{R}$ and:

(1) if $(X, E) \in \mathcal{R}$ then $(E', E) \in \mathcal{R}$ with $X \overset{\text{def}}{=} E'$;
(2) if $(E, X) \in \mathcal{R}$ then $(E, E') \in \mathcal{R}$ with $X \overset{\text{def}}{=} E'$;
(3) if $(\langle E_i \rangle_{i \in I}, E) \in \mathcal{R}$ then $\forall i \in I.\ (E_i, E) \in \mathcal{R}$;
(4) if $(E, \langle E_i \rangle_{i \in I}) \in \mathcal{R}$ then $\forall i \in I.\ (E, E_i) \in \mathcal{R}$;
(5) if $(\langle a_i : E_i \rangle_{i \in I}, \langle a_j : E_j' \rangle_{j \in J}) \in \mathcal{R}$ then $I \subseteq J$ and $\forall i \in I.\ (E_i, E_i') \in \mathcal{R}$.

We extend the use of $\sqsubseteq$, defined on input tree equations, to terms $E \in T_{\mathcal{V}}$ and $E' \in T'_{\mathcal{V}}$; namely, we write $E \sqsubseteq E'$ if there exists a compatibility relation $\mathcal{R}$ s.t. $(E, E') \in \mathcal{R}$.

Notice that compatibility is formally defined following a coinductive approach that performs the following checks on $\mathcal{G}$ and $\mathcal{G}'$, starting from the initial pair (X_0, X_0'). The first two items of Definition 3.12 let variables be replaced by their respective definitions. The next two items explore all the successors of silent choices. The last item guarantees that all the receive actions of the l.h.s. can be actually matched by receive actions in the r.h.s. The check of compatibility will be used in Definition 3.16, in order to control that the candidate supertype always has input branchings included in those of the candidate subtype. More precisely, we will check that the system of input tree equations, that represents the possible inputs of the supertype, is compatible with that of the candidate subtype.

Example 3.13. Consider the two systems of input tree equations in Figure 5. We have $\mathcal{G} \sqsubseteq \mathcal{G}'$. We enumerate a few pairs which must be in the embedding relation:

Initial variables:	(X_0, Y_{n_8})
Unfold X_0 (Case (1) of Def. 3.12)	$(\langle ok : X_{q_2,n_8}, ko : X_{q_2,n_8} \rangle, Y_{n_8})$
Unfold Y_{n_8} (Case (2) of Def. 3.12)	$(\langle ok : X_{q_2,n_8}, ko : X_{q_2,n_8} \rangle, \langle Y_{n_9} \rangle)$
Silent-choice (Case (4) of Def. 3.12)	$(\langle ok : X_{q_2,n_8}, ko : X_{q_2,n_8} \rangle, Y_{n_9})$
Unfold Y_{n_9} (Case (2) of Def. 3.12)	$(\langle ok : X_{q_2,n_8}, ko : X_{q_2,n_8} \rangle, \langle ok : Y_{n_8},\ ko : Y_{n_{10}} \rangle)$
Choice (Case (2) of Def. 3.12)	(X_{q_2,n_8}, Y_{n_8}) and $(X_{q_2,n_8}, Y_{n_{10}})$

$$\begin{aligned}
X_0 &\overset{\text{def}}{=} \langle ok : X_{q_2,n_8}, ko : X_{q_2,n_8}\rangle \\
X_{q_2,n_8} &\overset{\text{def}}{=} \langle ok : X_{q_2,n_9},\ ko : X_{q_2,n_9}\rangle \\
X_{q_2,n_9} &\overset{\text{def}}{=} \langle X_{q_2,n_8},\ X_{q_2,n_{10}}\rangle \\
X_{q_2,n_{10}} &\overset{\text{def}}{=} \langle ok : X_{q_2,n_{12}},\ ko : X_{q_2,n_{12}}\rangle \\
X_{q_2,n_{12}} &\overset{\text{def}}{=} \langle ok : X_{q_2,n_{13}},\ ko : X_{q_2,n_{13}}\rangle \\
X_{q_2,n_{13}} &\overset{\text{def}}{=} \langle X_{q_2,n_{12}},\ X_{q_2,n_{15}}\rangle \\
X_{q_2,n_{15}} &\overset{\text{def}}{=} \langle ok : X_{q_2,n_8},\ ko : X_{q_2,n_8}\rangle
\end{aligned}$$

$$\begin{aligned}
Y_{n_8} &\overset{\text{def}}{=} \langle Y_{n_9}\rangle \\
Y_{n_9} &\overset{\text{def}}{=} \langle ok : Y_{n_8},\ ko : Y_{n_{10}}\rangle \\
Y_{n_{10}} &\overset{\text{def}}{=} \langle Y_{n_{12}}\rangle \\
Y_{n_{13}} &\overset{\text{def}}{=} \langle ok : Y_{n_{12}},\ ko : Y_{n_{15}}\rangle \\
Y_{n_{15}} &\overset{\text{def}}{=} \langle Y_{n_8}\rangle
\end{aligned}$$

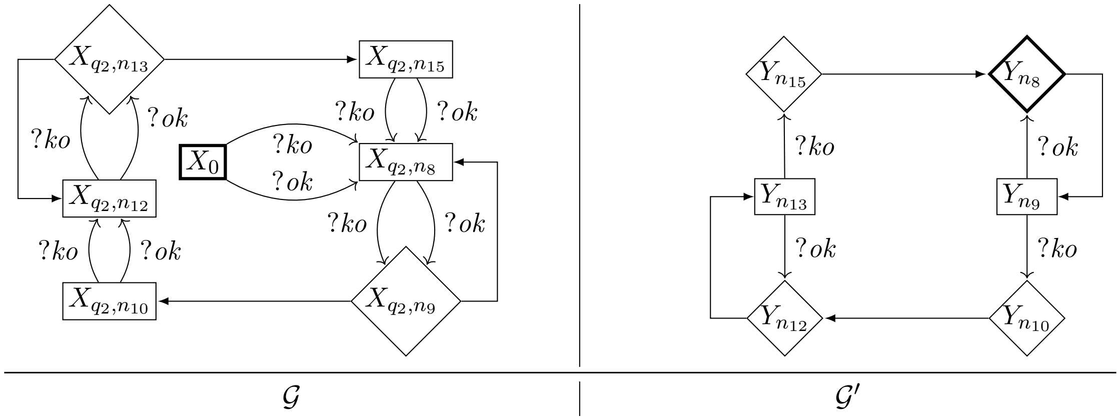


Figure 5: Input tree equations for $M_R \preccurlyeq M_C$ (Figures 1 and 2) and their graphical representations. The starting variables are X_0 and Y_{n_8}. Silent choices are diamond-shaped nodes, other nodes are rectangles.

Before giving the definition of a witness subtree, we introduce a few auxiliary functions on which it relies. Given a machine $M = (Q, q_0, \delta)$, a state $q \in Q$, and a word $\omega \in \mathbb{A}^*$, we define $\mathsf{accTree}(q, \omega)$ as follows:

$$\mathsf{accTree}(q,\omega) = \begin{cases} q & \text{if } \omega = \epsilon \\ \mathcal{A}[\mathsf{accTree}(q_i', \omega')]^{i\in I} & \text{if } \omega = a \cdot \omega', \mathcal{A}[q_i]^{i\in I} = \mathsf{inTree}(q), \forall i \in I.\ q_i \xrightarrow{!a} q_i' \\ \bot & \text{otherwise} \end{cases}$$

Function $\mathsf{accTree}(q, \omega)$ is a key ingredient of the witness subtree definition as it allows for the construction of the accumulation of receive actions (represented as an input tree) that is generated from a state q mimicking the sequence of send actions sending the messages in ω. We illustrate the usage of $\mathsf{accTree}(q, \omega)$ in Example 3.14 below.

We use the auxiliary function $\mathsf{minAcc}(k, Q', \psi)$ below to ensure that the effect of performing the transitions from an ancestor to a boundary node is that of increasing (possibly non-strictly) the accumulated receive actions. Here, k represents a known lower bound for the length of the sequences of receive actions accumulated in an input context $\mathcal{A}$, i.e., a lower bound for $\mathsf{minHeight}(\mathcal{A})$. Assuming that the holes in $\mathcal{A}$ contain the states populating the set of states Q', the function returns a lower bound for the length of the sequences of accumulated receive actions after the transitions in ψ have been executed. Formally, given a natural number k ($k \geq 0$), a sequence of action $\psi \in Act^*$, and a set of states $\{q_j \mid j \in J\} \subseteq Q$, we define this function as follows:

$$\mathsf{minAcc}(k, \{q_j \mid j \in J\}, \psi) =$$

$$\begin{cases} k & \text{if } \psi = \epsilon \\ \mathsf{minAcc}(k-1, \{q_j \mid j \in J\}, \psi') & \text{if } \psi = ?a \cdot \psi' \wedge k > 0 \\ \mathsf{minAcc}(k + \min_{j \in J} \mathsf{minHeight}(\mathcal{A}_j), \{q_{i,j} | j \in J, i \in I_j\}, \psi') & \begin{array}{l}\text{if } \psi = !a \cdot \psi' \wedge \\ \forall j \in J.\, \mathsf{accTree}(q_j, a) = \mathcal{A}_j[q_{i,j}]^{i \in I_j}\end{array} \\ \bot & \text{otherwise} \end{cases}$$

Example 3.14. Consider the transitions from node n_7 to n_9 in Figure 4. There are two send actions $!pr$ and $!nd$ that cannot be directly fired from state q_2 which is a receiving state; the effect is to accumulate receive actions. Such an accumulation is computed by $\mathsf{accTree}(q_2, pr \cdot nd) = \langle ko : \langle ko : q_2, ok : q_2 \rangle, ok : \langle ko : q_2, ok : q_2 \rangle \rangle$. For this sequence of transitions, the effect on the (minimal) length of the accumulated receive actions can be computed by $\mathsf{minAcc}(0, \{q_2\}, !pr \cdot !nd) = 2$; meaning that before executing the sequence of transitions $!pr \cdot !nd$ state q_2 has not accumulated receive actions in front, while at the end an input context with minimal depth 2 is generated as accumulation.

We now prove a couple of properties of $\mathsf{minAcc}(n, Q', \psi)$.

Proposition 3.15. *If $\mathsf{minAcc}(k, Q', \psi)$ is defined, then the following statements hold:*

(1) *for each $k' \geq k$ we have that $k - \mathsf{minAcc}(k, Q', \psi) = k' - \mathsf{minAcc}(k', Q', \psi)$;*
(2) *if $\psi = \psi' \cdot \psi''$ then $\mathsf{minAcc}(k, Q', \psi' \cdot \psi'') = \mathsf{minAcc}(\mathsf{minAcc}(k, Q', \psi'), Q'', \psi'')$ with $Q'' = \bigcup_{q \in Q'} \{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q, \mathsf{snd}(\psi')) = \mathcal{A}''[q_h]^{h \in H}\}$;*
(3) *if $\mathsf{minAcc}(k, Q'', \psi)$ is defined for a set of states Q'' s.t. $Q' \subseteq Q''$ then $\mathsf{minAcc}(k, Q'', \psi) \leq \mathsf{minAcc}(k, Q', \psi)$.*

A direct consequence of (1) *is that:* $\mathsf{minAcc}(k', Q', \psi) - \mathsf{minAcc}(k, Q', \psi) = k' - k \geq 0$.

Proof. Item 1 is proved by induction on the length of ψ. If the length of ψ is 0 then $\psi = \epsilon$ and $k - \mathsf{minAcc}(k, Q', \epsilon) = k' - \mathsf{minAcc}(k', Q', \epsilon) = 0$. In the inductive case we have two distinct cases: if $\psi = ?a \cdot \psi'$ we have $k > 0$ and $k - \mathsf{minAcc}(k, Q', ?a \cdot \psi') = k - \mathsf{minAcc}(k-1, Q', \psi')$ and $k' > 0$ and $k' - \mathsf{minAcc}(k', Q', ?a \cdot \psi') = k' - \mathsf{minAcc}(k'-1, Q', \psi')$, and by inductive hypothesis $k - 1 - \mathsf{minAcc}(k-1, Q', \psi') = k' - 1 - \mathsf{minAcc}(k'-1, Q', \psi')$ hence also $k - \mathsf{minAcc}(k-1, Q', \psi') = k' - \mathsf{minAcc}(k'-1, Q', \psi')$; if $\psi = !a \cdot \psi'$ we have $k - \mathsf{minAcc}(k, Q', !a \cdot \psi') = k - \mathsf{minAcc}(k + w, Q'', \psi')$ and $k' - \mathsf{minAcc}(k', Q', !a \cdot \psi') = k' - \mathsf{minAcc}(k' + w, Q'', \psi')$ for a set of states Q'' and a value w, and by inductive hypothesis $k + w - \mathsf{minAcc}(k + w, Q'', \psi') = k' + w - \mathsf{minAcc}(k' + w, Q'', \psi')$ hence also $k - \mathsf{minAcc}(k + w, Q'', \psi') = k' - \mathsf{minAcc}(k' + w, Q'', \psi')$.

Item 2 is proved by induction on the length of ψ'. In the base case, the thesis directly follows from $\mathsf{minAcc}(k, Q', \epsilon) = k$ and $\mathsf{accTree}(q, \epsilon) = q$ (hence we have $Q'' = Q'$). In the inductive case we have two distinct cases: If $\psi' = ?a \cdot \psi'''$ we have ($k > 0$ because $\mathsf{minAcc}(k, Q', ?a \cdot \psi''' \cdot \psi'')$ is defined) $\mathsf{minAcc}(\mathsf{minAcc}(k, Q', ?a \cdot \psi'''), Q'', \psi'') = \mathsf{minAcc}(\mathsf{minAcc}(k-1, Q', \psi'''), Q'', \psi'')$, but the latter, by applying the inductive hypothesis, coincides with $\mathsf{minAcc}(k-1, Q', \psi''' \cdot \psi'') = \mathsf{minAcc}(k, Q', ?a \cdot \psi''' \cdot \psi'')$. If $\psi' = !a \cdot \psi'''$ we have $\mathsf{minAcc}(\mathsf{minAcc}(k, Q', !a \cdot \psi'''), Q'', \psi'') = \mathsf{minAcc}(\mathsf{minAcc}(k + w, Q''', \psi'''), Q'', \psi'')$, for the set of states Q''' obtained by allowing the states in Q' to anticipate $!a$ and a value w that depends on Q' and a (see definition of the minAcc function); the latter, by applying the inductive hypothesis, coincides with $\mathsf{minAcc}(k + w, Q''', \psi''' \cdot \psi'')$ (because Q'' is obtained from Q''' by allowing the states in Q''' to anticipate the send actions in ψ'''), which is equal to $\mathsf{minAcc}(k, Q', !a \cdot \psi''' \cdot \psi'')$ as Q''' and w depend on Q' and a as described above.

Item 3 is proved by induction on the length of ψ. The base case is trivial because $\mathsf{minAcc}(k, Q'', \epsilon) = \mathsf{minAcc}(k, Q', \epsilon) = k$. In the inductive case we have two distinct cases: If $\psi = ?a \cdot \psi'$ we have $\mathsf{minAcc}(k, Q'', ?a \cdot \psi') = \mathsf{minAcc}(k-1, Q'', \psi')$ with the latter, by inductive hypothesis, that is smaller than or equal to $\mathsf{minAcc}(k-1, Q', \psi') = \mathsf{minAcc}(k, Q', ?a \cdot \psi')$. If $\psi = !a \cdot \psi'$ we have $\mathsf{minAcc}(k, Q'', !a \cdot \psi') = \mathsf{minAcc}(k+w, Q''', \psi')$ for a set of states Q''' obtained from Q'' by allowing its states to anticipate $!a$ and w the corresponding minimal height. Now, if we allow the states in the smaller (or equal) set Q' to anticipate $!a$, we obtain a smaller (or equal) set Q'''' and a value w' that cannot be smaller than w, hence we can apply the inductive hypothesis to obtain the greater or equal value $\mathsf{minAcc}(k + w, Q'''', \psi')$, which is smaller or equal, for item 1 of this Proposition, than $\mathsf{minAcc}(k+w', Q'''', \psi') = \mathsf{minAcc}(k, Q', !a \cdot \psi')$. □

We finally give the definition of witness subtree.

Definition 3.16 (Witness Subtree)**.** Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$. A candidate subtree of $\mathsf{simtree}(M_1, M_2)$ with root r, boundary B, and ancestor function anc, is a *witness* if the following holds:

(1) For all $n \in B$, given ψ such that $\mathsf{anc}(n) \overset{\psi}{\hookrightarrow} n$, we have $|\mathsf{rcv}(\psi)| > 0$.

(2) For all $n \in img(\mathsf{anc})$ and $n' \in img(\mathsf{anc}) \cup B$ such that $n \overset{\psi}{\hookrightarrow} n'$, $\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i \in I}$, and $\mathcal{L}(n') = p' \preccurlyeq \mathcal{A}'[q_j]^{j \in J}$, we have that :
   (a) $\forall i \in I \,.\, \{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \mathsf{snd}(\psi)) = \mathcal{A}''[q_h]^{h \in H}\} \subseteq \{q_j \mid j \in J\}$;
   (b) if $n' \in B$ then $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi) \geq \mathsf{minHeight}(\mathcal{A})$.

(3) $\mathcal{G} \sqsubseteq \mathcal{G}'$ where
   (a) $\mathcal{G} = (\{X_0\} \cup \{X_{q,n} \mid q \in Q, n \in \mathsf{nodes}(S, r, B) \backslash B\}, X_0, \mathbf{E})$ with $\mathbf{E}$ defined as follows:
      (i) $X_0 \overset{\text{def}}{=} T\{X_{q,r}/q \mid q \in Q\}$, with $\mathcal{L}(r) = p \preccurlyeq T$
      (ii) $X_{q,n} \overset{\text{def}}{=}$

$$\begin{cases} \langle X_{q,tr(n')} \mid \exists a.n \overset{?a}{\hookrightarrow} n' \rangle & \text{if } \exists a.n \overset{?a}{\hookrightarrow} \\ \langle \mathcal{A}[X_{q'_i,tr(n')}]^{i \in I} \mid \exists a.n \overset{!a}{\hookrightarrow} n' \wedge \mathsf{inTree}(q) = \mathcal{A}[q_i]^{i \in I} \wedge \forall i \in I.q_i \xrightarrow{!a} q'_i \rangle & \text{otherwise} \end{cases}$$

   (b) $\mathcal{G}' = (\{Y_n \mid n \in \mathsf{nodes}(S, r, B) \backslash B\}, Y_r, \mathbf{E}')$ with $\mathbf{E}'$ defined as follows:

$$Y_n \overset{\text{def}}{=} \begin{cases} \langle Y_{tr(n')} \mid n \overset{!a}{\hookrightarrow} n' \rangle & \text{if } \exists n'.n \overset{!a}{\hookrightarrow} n' \\ \langle a : Y_{tr(n')} \mid n \overset{?a}{\hookrightarrow} n' \rangle & \text{if } \exists n'.n \overset{?a}{\hookrightarrow} n' \end{cases}$$

   where $tr(n) = n$, if $n \notin B$; $tr(n) = \mathsf{anc}(n)$, otherwise.

Condition (1) requires the existence of a receive transition between an ancestor and a boundary node. This implies that if the behaviour beyond the witness subtree is the repetition of behaviour already observed in the subtree, then there cannot be send-only cycles.

Condition (2a) requires that the transitions from ancestors to boundary nodes (or to other ancestors) are such that they include those behaviours that can be computed by the $\mathsf{accTree}$ function. We assume that this condition does not hold if $\mathsf{accTree}(q_i, \mathsf{snd}(\psi)) = \bot$ for any $i \in I$; hence the states q_i of M_2 in an ancestor are able to mimic all the send actions performed by M_1 along the sequences of transitions in the witness subtree starting from the considered ancestor.

Condition (2b) ensures that by repeating transitions from ancestors to boundary nodes, the accumulation of receive actions is, overall, non-decreasing. In other words, the rate at which accumulation is taking place is higher than the rate at which the context is reduced by Rule (InCtx).
Condition (3) checks that the receive actions that can be accumulated by M_2(represented by $\mathcal{G}$) and those that are expected to be actually executed by M_1 (represented by $\mathcal{G}'$) are compatible. In $\mathcal{G}$, there is an equation for the root node and for each pair consisting of a local state in M_2 and a node n in the witness subtree. The equation for the root node is given in (3(a)i), where we simply transform an input context into an input tree expression. The other equations are given in (3(a)ii), where we use the partial function $\mathsf{inTree}(q)$. Each equation represents what can be accumulated by starting from node n (focusing on local state q). In $\mathcal{G}'$, there is an equation for each node n in the witness subtree, as defined in (3b) There are two types of equations depending on the type of transitions outgoing from node n. A send transition leads to silent choices, while receive transitions generate corresponding receive choices.

Example 3.17. We have that the candidate subtree rooted at n_8 in Figure 4 satisfies Definition 4. (1) Each path from an ancestor to a boundary node includes at least one receive action. (2a) For each sequence of transitions from an ancestor to a boundary node (or another ancestor) the behaviour of the states of M_2, as computed by the $\mathsf{accTree}$ function, has already been observed. (2b) For each sequence of transitions from an ancestor to a boundary node, the rate at which receive actions are accumulated is higher than or equal to the rate at which they are removed from the accumulation. (3) The systems of input tree equations $\mathcal{G}$ (3a) and $\mathcal{G}'$ (3b) are given in Figure 5, and are compatible, see Example 3.13.

We now describe how $\mathcal{G}$ and $\mathcal{G}'$ (Figure 5) are constructed from the witness tree rooted at n_8 in Figure 4. For $\mathcal{G}$ we have the following equations:

- $X_0 \overset{\text{def}}{=} \langle ok : X_{q_2,n_8}, ko : X_{q_2,n_8}\rangle$ since the root of the witness tree is n_8 and its label is $q_1 \preccurlyeq \langle ok : q_2,\ ko : q_2\rangle$. In Figure 5, we depict this equation as a pair of transitions from the node labelled by X_0 to the node labelled by X_{q_2,n_8}
- $X_{q_2,n_8} \overset{\text{def}}{=} \langle ok : X_{q_2,n_9},\ ko : X_{q_2,n_9}\rangle$ since n_8 has a unique outgoing *send* transition to n_9, i.e., n' in Case (3(a)ii) of Definition 3.16, and $\mathsf{inTree}(q_2) = \langle ok : q_1,\ ko : q_1\rangle$ with $q_1 \xrightarrow{!nd} q_2$ and $q_1 \xrightarrow{!pr} q_2$ in M_C
- $X_{q_2,n_9} \overset{\text{def}}{=} \langle X_{q_2,n_8},\ X_{q_2,n_{10}}\rangle$ since n_9 has two *receive* transitions: one to n_{11} (a boundary node whose ancestor is n_8, i.e., $tr(n_{11}) = n_8$) and one to n_{10} (which is not boundary node, i.e., $tr(n_{10}) = n_{10}$)
- $X_{q_2,n_{10}} \overset{\text{def}}{=} \langle ok : X_{q_2,n_{12}},\ ko : X_{q_2,n_{12}}\rangle$ since n_{10} has a unique outgoing *send* transition to n_{12}, and $\mathsf{inTree}(q_2) = \langle ok : q_1,\ ko : q_1\rangle$
- $X_{q_2,n_{12}} \overset{\text{def}}{=} \langle ok : X_{q_2,n_{13}},\ ko : X_{q_2,n_{13}}\rangle$ since n_{12} has a unique outgoing *send* transition to n_{13}, and $\mathsf{inTree}(q_2) = \langle ok : q_1,\ ko : q_1\rangle$
- $X_{q_2,n_{13}} \overset{\text{def}}{=} \langle X_{q_2,n_{12}},\ X_{q_2,n_{15}}\rangle$ since n_9 has two *receive* transitions: one to n_{14} (a boundary node whose ancestor is n_{12}) and one to n_{15} (which is not boundary node)
- $X_{q_2,n_{15}} \overset{\text{def}}{=} \langle ok : X_{q_2,n_8},\ ko : X_{q_2,n_8}\rangle$ since $\mathsf{inTree}(q_2) = \langle ok : q_1,\ ko : q_1\rangle$ and n_{15} has a unique outgoing *send* transition to n_{16}, which is a boundary node ($n_{16} \in B$) whose ancestor is n_8.

We omit the other equations, e.g., X_{q_1,n_8} as they are not reachable from X_0.

For $\mathcal{G}'$ we have the following equations:

- $Y_{n_8} \overset{\text{def}}{=} \langle Y_{n_9} \rangle$ since n_8 has a unique *send* transition to n_9, i.e., n' in Case (3b) of Definition 3.16, and n_9 is not a boundary node
- $Y_{n_9} \overset{\text{def}}{=} \langle ok : Y_{n_8},\ ko : Y_{n_{10}} \rangle$ since n_9 has two *receive* transitions: one to n_{10} which is not a boundary node, and one to n_{11} which is a boundary node whose ancestor is n_8
- $Y_{n_{10}} \overset{\text{def}}{=} \langle Y_{n_{12}} \rangle$ since n_{10} has a unique *send* transition to n_{12} which is not a boundary node
- $Y_{n_{12}} \overset{\text{def}}{=} \langle Y_{n_{13}} \rangle$ since n_{12} has a unique *send* transition to n_{13} which is not a boundary node
- $Y_{n_{13}} \overset{\text{def}}{=} \langle ok : Y_{n_{12}},\ ko : Y_{n_{15}} \rangle$ since n_{13} has two *receive* transitions: one to n_{15} which is not a boundary node, and one to n_{14} which is a boundary node whose ancestor is n_{12}
- $Y_{n_{15}} \overset{\text{def}}{=} \langle Y_{n_8} \rangle$ since n_{15} has a unique *send* transition to n_{16}, and n_{16} is a boundary node whose ancestor is n_8.

We now prove the main property of the $\mathsf{minAcc}(k, Q', \psi)$ function, i.e., given information k and Q' extracted from an ancestor n in a witness subtree, such a function correctly computes a lower bound of the length of the input accumulation in a node n' reachable from n by executing the sequence of actions ψ.

Proposition 3.18. *Consider a witness subtree with ancestor function* anc*; given two nodes of the tree,* $n \in img(\mathsf{anc})$ *and* n' *s.t.* $n \overset{\psi}{\hookrightarrow} n'$*, with* $\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i \in I}$ *and* $\mathcal{L}(n') = p' \preccurlyeq \mathcal{A}'[q_j]^{j \in J}$*, we have that* $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi) \leq \mathsf{minHeight}(\mathcal{A}')$*.*

Proof. We prove a more general result proceeding by induction on the length of ψ, i.e., that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi) \leq \mathsf{minHeight}(\mathcal{A}')$ and $\{q_j \mid j \in J\} \subseteq \bigcup_{i \in I}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h \in H}\}$.

The base case is trivial because, by definition, $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \epsilon) = \mathsf{minHeight}(\mathcal{A})$ and having $n = n'$ then $\mathsf{minHeight}(\mathcal{A}) = \mathsf{minHeight}(\mathcal{A}')$. Moreover, $\bigcup_{i \in I}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \epsilon) = \mathcal{A}'''[q_h]^{h \in H}\} = \{q_i \mid i \in I\}$ and having $n = n'$ then $\{q_i \mid i \in I\} = \{q_j \mid j \in J\}$.

In the inductive case we have either $\psi = \psi' \cdot ?a$ or $\psi = \psi' \cdot !a$. In both cases we observe that, by definition of witness subtree, $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi)$ is defined as it is defined for a longer sequence of transitions from n to a boundary node (traversing n').

We first consider $\psi = \psi' \cdot ?a$. Let n'' be the node reached after the sequence of transitions ψ', and let $\mathcal{L}(n'') = p'' \preccurlyeq \mathcal{A}''[q_w]^{w \in W}$. By inductive hypothesis we have that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi') \leq \mathsf{minHeight}(\mathcal{A}'')$ and, letting $Q'' = \bigcup_{i \in I}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \mathsf{snd}(\psi')) = \mathcal{A}'''[q_h]^{h \in H}\}$, we also have $\{q_w \mid w \in W\} \subseteq Q''$. By Proposition 3.15, item 2, we have that the following holds: $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi' \cdot ?a) = \mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi'), Q'', ?a)$. By definition of minAcc, we also have that $\mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi'), Q'', ?a) = \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi') - 1$. As a direct consequence of the inductive hypothesis we have $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi') - 1 \leq \mathsf{minHeight}(\mathcal{A}'') - 1$, but we have that $\mathsf{minHeight}(\mathcal{A}'') - 1 \leq \mathsf{minHeight}(\mathcal{A}')$, because the effect of an input transition on the input context is simply that of consuming one initial input branching. We conclude this case by observing that $\{q_j \mid j \in J\} \subseteq \{q_w \mid w \in W\}$ because, as observed above, the effect of an input transition on the input context is simply that of consuming one initial input branching, without changing the states populating the leaves of the input tree. On the other hand, the set of states obtained from the states q_i by anticipating the outputs in $\psi' \cdot ?a$ coincides with the above set Q'' because only send actions are considered, and $\mathsf{snd}(\psi') = \mathsf{snd}(\psi'\cdot?a)$. By inductive hypothesis, we have $\{q_w \mid w \in W\} \subseteq Q''$.

We now consider $\psi = \psi'\cdot!a$. Let n'' be the node reached after the sequence of transitions ψ', and let $\mathcal{L}(n'') = p'' \preccurlyeq \mathcal{A}''[q_w]^{w\in W}$. By inductive hypothesis we have that

$$\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi') \leq \mathsf{minHeight}(\mathcal{A}'')$$

and, letting

$$Q'' = \bigcup_{i\in I}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \psi') = \mathcal{A}'''[q_h]^{h\in H}\}$$

we also have $\{q_w \mid w \in W\} \subseteq Q''$. By Proposition 3.15, item 2,

$$\begin{aligned}&\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi'\cdot!a)\\ &= \mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi'), Q'', !a)\end{aligned}$$

moreover, by definition of minAcc, we have

$$\begin{aligned}&\mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi'), Q'', !a)\\ &= \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi') + z\end{aligned}$$

with z the minimal depth of the holes in the input tree that are accumulated by the states in Q'' when they anticipate $!a$. Having $n'' \overset{!a}{\hookrightarrow} n'$, we have that the minimal depth of the input tree in n', i.e. $\mathsf{minHeight}(\mathcal{A}')$, will increase that of n'', i.e. $\mathsf{minHeight}(\mathcal{A}'')$, depending on new accumulation generated by the anticipation of $!a$, hence the increase, i.e. $\mathsf{minHeight}(\mathcal{A}') - \mathsf{minHeight}(\mathcal{A}'')$, will be greater than or equal to the minimal depth of the holes in the input tree that are accumulated by the states in $\{q_w \mid w \in W\}$ when they anticipate $!a$. Being $\{q_w \mid w \in W\} \subseteq Q''$, we have that such an increase will be also greater than or equal to z, i.e. $\mathsf{minHeight}(\mathcal{A}') - \mathsf{minHeight}(\mathcal{A}'') \geq z$. As a direct consequence of the inductive hypothesis we have

$$\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), \{q_i \mid i \in I\}, \psi') + z \leq \mathsf{minHeight}(\mathcal{A}'') + z \leq \mathsf{minHeight}(\mathcal{A}')$$

We now consider $Q''' = \bigcup_{i\in I}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \psi'\cdot!a) = \mathcal{A}'''[q_h]^{h\in H}\}$; we have that the states in Q''' are generated by the states in Q'' when they anticipate the output $!a$. The same holds also for $\{q_j \mid j \in J\}$, i.e., the states q_j are generated by the states in $\{q_w \mid w \in W\}$ when they anticipate the output $!a$. Having $\{q_w \mid w \in W\} \subseteq Q''$, we also have $\{q_j \mid j \in J\} \subseteq Q'''$. □

We conclude by proving our main result; given a simulation tree with a witness subtree with root r, all the branches in the simulation tree traversing r are infinite (hence successful).

Theorem 3.19. *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with* $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$*. If* $\mathsf{simtree}(M_1, M_2)$ *has a witness subtree with root r then for every node $n \in N$ such that $r \hookrightarrow^* n$ there exists n' such that $n \hookrightarrow n'$.*

Proof. Let B be the leaves of the witness subtree rooted in r (i.e. the witness subtree is $\mathsf{nodes}(S, r, B)$). If there exists $l \in B$ such that $n \hookrightarrow^+ l$ the thesis trivially holds. For all other nodes n such that $r \hookrightarrow^* n$, there exists $l \in B$ such that $l \hookrightarrow^* n$.

We now prove by induction on the length of $l \hookrightarrow^* n$, with $\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i\in I}$, that there exist $m, m' \in \mathsf{nodes}(S, r, B) \setminus B$, s.t. $m \in \mathit{img}(\mathsf{anc})$, $m \overset{\psi}{\hookrightarrow} m'$, $\mathcal{L}(m) = p' \preccurlyeq \mathcal{A}'[q_j]^{j\in J}$, $\mathcal{L}(m') = p \preccurlyeq \mathcal{A}''[q_k]^{k\in K}$ such that:

- $\{q_i \mid i \in I\} \subseteq \bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h\in H}\}$;

- $\mathcal{A}[X_{q_i,m'}]^{i\in I} \sqsubseteq Y_{m'}$;
- $\mathsf{minHeight}(\mathcal{A}) \geq \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi)$.

The base case is when $n \in B$. In this case, let $m, m' = \mathsf{anc}(n)$.

The first item follows from the definition of candidate subtree according to which $\{q_i \mid i \in I\} \subseteq \{q_j \mid j \in J\}$.

The second item follows from the following reasoning: we consider $X_0 \sqsubseteq Y_r$ and apply on such pair the following transformations of the l.h.s. X_0 and r.h.s. Y_r. We consider the sequence of transitions from r to n and proceed as follows. For each receive transition $o \overset{?a}{\hookrightarrow} o'$ we modify the r.h.s. by considering $Y_{tr(o')}$ and the l.h.s. by consuming the initial message a and by replacing each variable $X_{q,o}$ (for any q) with the variable $X_{q,tr(o')}$ that, inside their corresponding definitions, is present because of the transition $o \overset{?a}{\hookrightarrow} o'$. For each send transition $o \overset{!a}{\hookrightarrow} o'$ we modify the r.h.s. by considering $Y_{tr(o')}$ and the l.h.s. by replacing each variable with the term that, inside their corresponding definitions, is present because of the transition $o \overset{!a}{\hookrightarrow} o'$. Since $tr(n) = m'$, we obtain $\mathcal{A}[X_{q_i,m'}]^{i\in I} \sqsubseteq Y_{m'}$. Notice that the relation $\sqsubseteq$ actually holds because in the modification of the initial terms X_0 and Y_r s.t. $X_0 \sqsubseteq Y_r$ we follow the simulation game formalized in the Definition 3.12 of input tree compatibility: in the case of input transitions $o \overset{?a}{\hookrightarrow} o'$ we consume an initial a in both terms and resolve some silent choice in the l.h.s; in the case of output transitions $o \overset{!a}{\hookrightarrow} o'$ we resolve the initial silent choice in the r.h.s. while in the l.h.s. we replace variables with their definition and resolve the initial silent choice in such definitions.

The third item coincides with proving that $\mathsf{minHeight}(\mathcal{A}) \geq \mathsf{minHeight}(\mathcal{A}')$ because, having $m = m'$, the sequence ψ is empty in the expression $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi)$. By definition of witness subtree, we have that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi') \geq \mathsf{minHeight}(\mathcal{A}')$ for every sequence of transitions ψ' from m to a boundary node, hence also to n. By Proposition 3.18, if we consider the sequence of transitions ψ' from $\mathsf{anc}(n)$ to n, we have that $\mathsf{minHeight}(\mathcal{A}) \geq \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi')$, from which we conclude $\mathsf{minHeight}(\mathcal{A}) \geq \mathsf{minHeight}(\mathcal{A}')$.

We now move to the inductive case. Suppose, by inductive hypothesis, that the above three properties hold for n s.t. $l \hookrightarrow^{+} n$, and consider $n \hookrightarrow n'$. We separate the analysis in two parts, the case in which an output action $n \overset{!a}{\hookrightarrow} n'$ is executed, and the opposite case in which $n \overset{?a}{\hookrightarrow} n'$. We have to show that in both cases there exist two nodes $m_1, m_2 \in \mathsf{nodes}(S, r, B) \backslash B$ such that the three properties, defined for n, m, m', hold also for n', m_1, m_2, respectively.

We now consider $n \overset{!a}{\hookrightarrow} n'$. In this case we have that $p \xrightarrow{!a} p'$, hence also $m' \overset{!a}{\hookrightarrow} m''$.

We first consider the case in which $m'' \notin B$: in this case we take $m_1 = m$ and $m_2 = m''$.

The first item holds because; by inductive hypothesis we have $\{q_i \mid i \in I\} \subseteq \bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h\in H}\}$; by Definition 3.16 of witness subtree, item 2a, we have that all the above states q_h can anticipate the output action $!a$ because $\mathsf{accTree}(q_j, \mathsf{snd}(\psi'))$ is defined for a sequence of actions ψ', from m to a boundary node, that contains $\mathsf{snd}(\psi)\cdot !a$ as a prefix; and the states in $\{q_i \mid i \in I\}$ are modified by the transition $!a$ in the same way as the same states that are present also in the superset $\bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h\in H}\}$ change considering the longer sequence $\mathsf{snd}(\psi)\cdot !a$ instead of $\mathsf{snd}(\psi)$ only.

The second item holds because; by inductive hypothesis we have $\mathcal{A}[X_{q_i,m'}]^{i\in I} \sqsubseteq Y_{m'}$; the accumulated input tree in n' is obtained by replacing each of the variables in $\mathcal{A}[X_{q_i,m'}]^{i\in I}$ with the term that, inside their corresponding definitions, is present because $m' \overset{!a}{\hookrightarrow} m''$ and because, as observed above, each state q_i can anticipate the output action $!a$; the l.h.s. term obtained in this way (by simply replacing variables with their definition and resolving initial silent choices) continue to be in $\sqsubseteq$ relation with $Y_{m'}$ hence also with $Y_{m''}$ which is present in the definition of $Y_{m'}$ because $m' \overset{!a}{\hookrightarrow} m''$.

The third item holds because; if we take $k = \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi)$, by inductive hypothesis we have $\mathsf{minHeight}(\mathcal{A}) \geq k$; by Proposition 3.15, item 2, we have that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a) = \mathsf{minAcc}(k, Q, !a)$ with $Q = \bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h\in H}\}$ where J is the set of indices of the holes in the input context in the label of node m; by inductive hypothesis (first item) we have that $\{q_i \mid i \in I\} \subseteq Q$ where I is the set of indices of the holes in the input context in the label of node n; by definition of the minAcc function the increment $\mathsf{minAcc}(k, Q, !a) - k$ cannot be strictly greater than the increment of $\mathsf{minHeight}$ when the transition $!a$ is executed from n to n', because $\mathsf{minAcc}(k, Q, !a)$ considers the minimal accumulation generated by the states Q when anticipating $!a$ and, having $\{q_i \mid i \in I\} \subseteq Q$, such a minimal accumulation cannot be greater than the accumulation generated by the states q_i present in the leaves of the input tree of n. From the inductive hypothesis $\mathsf{minHeight}(\mathcal{A}) \geq k$ we, thus, have that $\mathsf{minHeight}$ in n' is greater or equal to $\mathsf{minAcc}(k, Q, !a)$.

We now consider the case in which $m'' \in B$. We have two distinct cases:

(1) $\mathsf{anc}(m'') \hookrightarrow^* m$
In this case we take $m_1 = m_2 = \mathsf{anc}(m'')$. The first item holds because of the same arguments considered in the corresponding case for $m'' \notin B$ plus the observation that $\bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi){\cdot}!a) = \mathcal{A}'''[q_h]^{h\in H}\}$ is a subset of the states in the holes of the input context in m'' (definition of witness subtree), which is a subset of the states in the holes of the input context in $\mathsf{anc}(m'')$ (definition of candidate subtree). The second item holds for the same argument considered in the case $m'' \notin B$ (simply replacing $Y_{m''}$ with $Y_{\mathsf{anc}(m'')}$). The third item holds for the following reasons. By applying the same arguments considered in the corresponding case for $m'' \notin B$ we obtain that the new $\mathsf{minHeight}$ in n' is greater or equal than $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a)$, where J is the set of indices of the holes in the input context in the label of node m; hence proving the third item reduces to prove that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a) \geq \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi')$, with $\mathcal{L}(m_1) = p_1 \preccurlyeq \mathcal{A}_1[q_w]^{w\in W}$, $Q_w = \{q_w \mid w \in W\}$ and ψ' corresponding to the sequence of transitions from $m_1 = \mathsf{anc}(m'')$ to m'' that traverses m, hence $\psi' = \psi'' \cdot \psi \cdot !a$ (for some ψ''). This is because $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi') \geq \mathsf{minHeight}(\mathcal{A}_1)$ by definition of witness subtree. By Proposition 3.15, item 2, we have $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi'' \cdot \psi{\cdot}!a) = \mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi''), Q'', \psi{\cdot}!a)$ with $Q'' = \bigcup_{q\in Q_w}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q, \mathsf{snd}(\psi'')) = \mathcal{A}_1[q_h]^{h\in H}\}$; given that the states $\{q_j \mid j \in J\}$ are generated starting from the states in Q_w by anticipation of the send actions in the sequence ψ'' we have that $\{q_j \mid j \in J\} \subseteq Q''$; by Proposition 3.15, item 3, we have that $\mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi''), Q'', \psi{\cdot}!a) \leq$
$\mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi''), \{q_j \mid j \in J\}, \psi{\cdot}!a)$; by Proposition 3.18 we have

that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi'') \leq \mathsf{minHeight}(\mathcal{A}')$ and as a consequence of Proposition 3.15, item 1, we have that $\mathsf{minAcc}(\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_1), Q_w, \psi''), Q'', \psi{\cdot}!a) \leq \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), Q'', \psi{\cdot}!a)$; finally by Proposition 3.15, item 3, we have $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), Q'', \psi{\cdot}!a) \leq \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a)$.

(2) $m \hookrightarrow^+ \mathsf{anc}(m'')$
In this case we take $m_1 = m$ and $m_2 = \mathsf{anc}(m'')$. The first item holds because of the same arguments considered in the corresponding case for $m'' \notin B$ plus the observation (as done in the previous case) that $\bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi){\cdot}!a) = \mathcal{A}'''[q_h]^{h\in H}\}$ is a subset of the states in the holes of the input context in m'' (definition of witness subtree); which is a subset of the states in the holes of the input context in $\mathsf{anc}(m'')$ (definition of candidate subtree); which is subset of $\bigcup_{j\in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi'')) = \mathcal{A}'''[q_h]^{h\in H}\}$ with ψ'' s.t. $m_1 = m \xhookrightarrow{\psi''} \mathsf{anc}(m'') = m_2$. Notice that the latter subset inclusion holds because the states in the holes of the input context in $\mathsf{anc}(m'') = m_2$ are generated starting from the states in Q_j by anticipation of the send actions in the sequence ψ''. The second item holds for the same arguments considered in the case $m'' \notin B$ (simply replacing $Y_{m''}$ with $Y_{\mathsf{anc}(m'')}$). We proceed by contraposition to show that the third item also holds. Given $\mathcal{L}(m) = p' \preccurlyeq \mathcal{A}'[q_j]^{j\in J}$ and $m \xhookrightarrow{\psi''} \mathsf{anc}(m'')$, we assume by contraposition that $\mathsf{minHeight}$ applied to n' is strictly smaller than $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi'')$. In the following we let $x = \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi'')$. By application of the same arguments as above (case $m'' \notin B$, third item), we have that $\mathsf{minHeight}$ applied to n' should be greater than or equal to $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a)$, hence also $x > \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a)$. But being m above $\mathsf{anc}(m'')$, we have that ψ'' is a prefix of ψ; then, by Proposition 3.15, item 2, we have $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi{\cdot}!a) = \mathsf{minAcc}(x, Q'', \psi'''{\cdot}!a)$ with $\psi = \psi'' \cdot \psi'''$ and $Q'' = \bigcup_{q\in Q_j}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q, \mathsf{snd}(\psi''')) = \mathcal{A}''[q_h]^{h\in H}\}$ where $Q_j = \{q_j \mid j \in J\}$. So far, we have proved that $x - \mathsf{minAcc}(x, Q'', \psi'''{\cdot}!a) > 0$. We now observe that, by Proposition 3.18, x is smaller than or equal to $\mathsf{minHeight}$ applied to $\mathsf{anc}(m'')$, i.e. assuming $\mathcal{L}(\mathsf{anc}(m'')) = p_w \preccurlyeq \mathcal{A}_2[q_w]^{w\in W}$ and $x' = \mathsf{minHeight}(\mathcal{A}_2)$, we have $x \leq x'$; by Proposition 3.15, item 2, we have that also $x' - \mathsf{minAcc}(x', Q'', \psi'''{\cdot}!a) > 0$, hence $x' > \mathsf{minAcc}(x', Q'', \psi'''{\cdot}!a)$. By definition of witness subtree, given that $m \in \mathit{img}(\mathsf{anc})$ and $m \xhookrightarrow{\psi''} \mathsf{anc}(m'')$, Q'' is a (non-strict) subset of the states $\{q_w \mid w \in W\}$, hence by Proposition 3.15, item 3, we obtain $\mathsf{minAcc}(x', Q'', \psi'''{\cdot}!a) \geq \mathsf{minAcc}(x', \{q_w \mid w \in W\}, \psi'''{\cdot}!a)$. By combination of the last two inequations we obtain $\mathsf{minHeight}(\mathcal{A}_2) > \mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}_2), \{q_w \mid w \in W\}, \psi'''{\cdot}!a)$ that contradicts the definition of witness subtree (item 2b).

We now consider $n \xhookrightarrow{?a} n'$. In this case we have that $p \xrightarrow{?a} p'$. We have that $\mathcal{A}$ cannot be a single hole, otherwise $\mathsf{minHeight}(\mathcal{A}) = 0$, that implies $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi) = 0$, that implies that there exists a sequence of transitions ψ', extending ψ and leading to a boundary node, such that $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}'), \{q_j \mid j \in J\}, \psi')$ is undefined, contrary to what definition of witness subtree says. Hence $\mathcal{A}[X_{q_i,m'}]^{i\in I}$ contains initially an a, that must be mimicked in the simulation game by $Y_{m'}$. This implies that also $m' \xhookrightarrow{?a} m''$. We first consider the case in which $m'' \notin B$: in this case we take $m_1 = m$ and $m_2 = m''$. The first item trivially holds because the set on the left cannot grow while the set on the right remains unchanged. The second item trivially holds because by inductive hypothesis we have $\mathcal{A}[X_{q_i,m'}]^{i \in I} \sqsubseteq Y_{m'}$; we modify the l.h.s. by consuming the initial inputs, taking the continuation of a, and replacing the remaining variables $X_{q_i,m'}$ with $X_{q_i,m''}$; as r.h.s. we take $Y_{m''}$. The relation $\sqsubseteq$ continue to hold as we follow on step a of the simulation game formalized in the Definition 3.12 of input tree compatibility, and we resolve some input choices in the l.h.s. The last item holds because the r.h.s. of the inequality reduce by one, while the l.h.s. cannot reduce by more than one. We now consider the case in which $m'' \in B$. There are two distinct cases: $\mathsf{anc}(m'') \hookrightarrow^* m$ or $m \hookrightarrow^+ \mathsf{anc}(m'')$. These two cases are treated as already done above for the case $n \overset{!a}{\hookrightarrow} n'$, subcase in which $m' \overset{!a}{\hookrightarrow} m''$ and $m'' \in B$.

We can finally prove the thesis considering $\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i \in I}$.

If p is sending, then m' can perform all send actions that p can do. Given any of such send actions $!a$, by definition of witness subtree we have that $\mathsf{accTree}(q_j, \mathsf{snd}(\psi'))$ is defined for a sequence of actions ψ', from m to a boundary node, that contains $\psi \cdot !a$ as a prefix; hence we have that all the states $\bigcup_{j \in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h \in H}\}$ can anticipate $!a$. Given that $\{q_i \mid i \in I\} \subseteq \bigcup_{j \in J}\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_j, \mathsf{snd}(\psi)) = \mathcal{A}'''[q_h]^{h \in H}\}$ we also have that all the states q_i can anticipate $!a$. The possibility to perform the transition $n \overset{!a}{\hookrightarrow} n'_a$ also requires that p has no infinite loop of send actions, i.e., $\neg\mathsf{cycle}(!, p)$. Assume by contraposition that p has such an infinite loop of send actions. This means that there exists an infinite sequence of output transitions in the witness subtree that starts from the node m' (which is such that $\mathcal{L}(m') = p \preccurlyeq \mathcal{A}''[q_k]^{k \in K}$) reaches a boundary node, and then continues from the ancestor of such boundary node to another boundary node, and so on. Eventually, an ancestor of a reached boundary node will be in between the last traversed ancestor and such boundary node (otherwise, we infinitely move strictly upward in the finite witness subtree, going from boundary nodes to ancestors that are always strictly above the last traversed ancestor). This contradicts the definition of witness subtree stating that in all paths from an ancestor $\mathsf{anc}(o)$, to a corresponding boundary node o, there is at least one receiving transition.

If p is receiving, then $\mathcal{A}$ cannot be a single hole (see the reasoning above for the case $n \overset{?a}{\hookrightarrow} n'$). Let $\mathcal{A} = \langle a_i : \mathcal{A}_i \rangle_{i \in I}$. Having $\mathcal{A}[X_{q_i,m'}]^{i \in I} \sqsubseteq Y_{m'}$, we have that (by definition of $Y_{m'}$ and $\sqsubseteq$), for every $i \in I$, there exists a transition $m' \overset{?a_i}{\hookrightarrow} m'_i$ hence also $p \xrightarrow{?a_i} p_i$. So we can conclude that we have also $n \overset{?a_i}{\hookrightarrow} n_i$, for every $i \in I$. □

Hence, we can conclude that if the candidate subtrees of $\mathsf{simtree}(M_1, M_2)$ identified following the strategy explained in Part (2) are also witness subtrees, then we have $M_1 \preccurlyeq M_2$.

Remark 3.20. When our algorithm finds a successful leaf, a previously seen label, or a witness subtree in each branch then the machines are in the subtyping relation. If an unsuccessful leaf is found (while generating the initial finite subtree as described in Part (2)), then the machines are *not* in the subtyping relation. In all other cases, the algorithm is unable to give a decisive verdict (i.e., the result is *unknown*). There are two possible causes for an unknown result: either (*i*) it is impossible to extract a forest of candidate subtrees (i.e., there are successful leaves below some ancestor) or (*ii*) at least one candidate subtree is not a witness (see Example 3.21).

Example 3.21. Consider the machines M_1 and M_2 below:

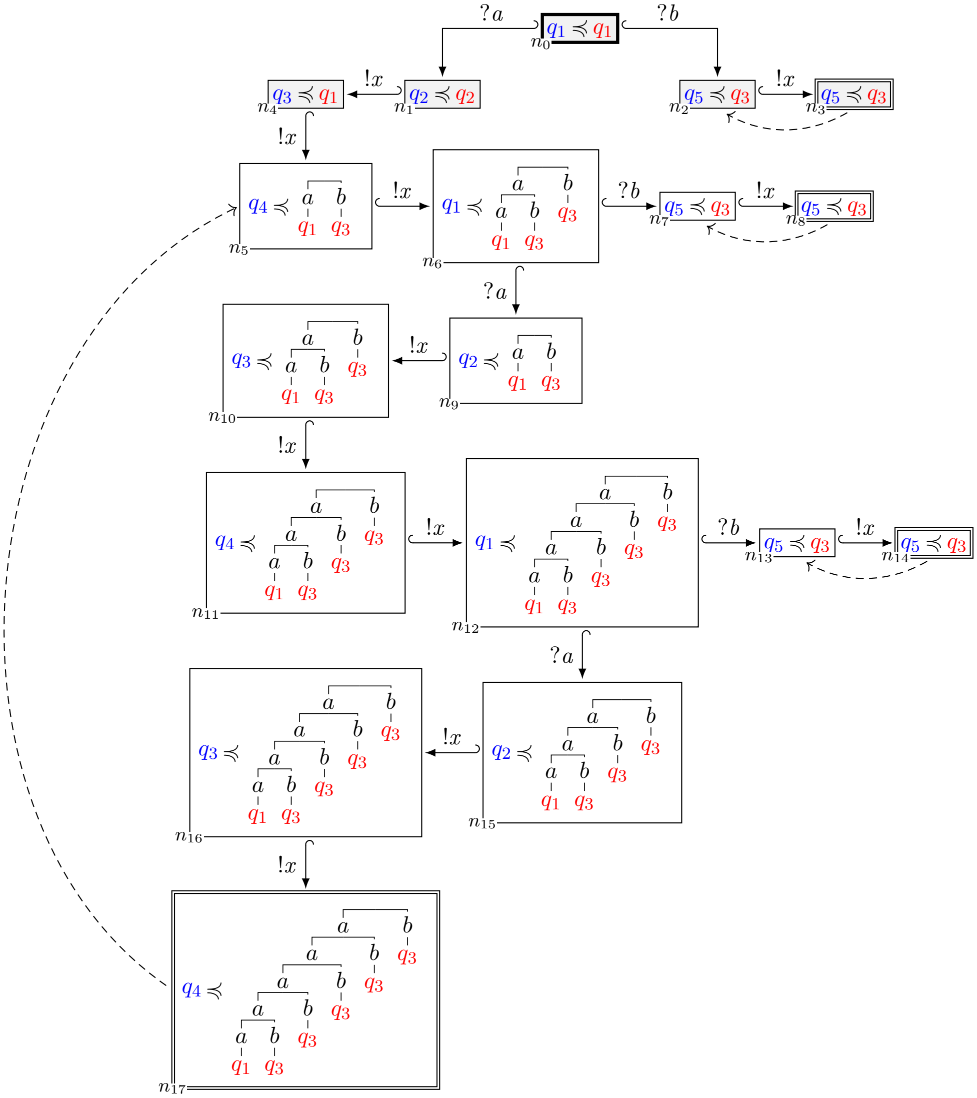
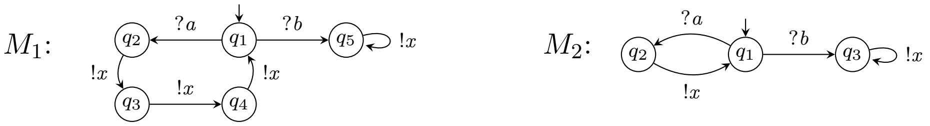


Figure 6: Simulation tree of Example 3.21.



The simulation tree $\mathsf{simtree}(M_1, M_2)$, whose initial part is given in Figure 6, contains infinitely many nodes with labels of the form: $q_1 \precsim \langle a : \langle a : \langle a : \langle \cdots \rangle, b : q_3 \rangle, b : q_3 \rangle, b : q_3 \rangle$ (e.g., n_6 and n_{12} in Figure 6). Each of these nodes has two successors, one where $?a$ is fired (the machines stay in their larger loops), and one where $?b$ is fired (the machines move to their self loops). The machines can always enter this send-only cycle, e.g., between n_2 and n_3 or between n_{13} and n_{14}. Because of these *send only* paths between ancestors (e.g., n_2) and leaves (e.g., n_3), Condition (1) of Definition 3.16 never applies on the infinite branches of

$\mathsf{simtree}(M_1, M_2)$, hence no witness subtrees can be found. Note however that our approach successfully identifies a candidate subtree, i.e., the white nodes in Figure 6.

4. Implementation and evaluation

To evaluate the applicability and cost of our algorithm, we have produced a faithful implementation of it, which is freely available on GitHub [BCL+19b].

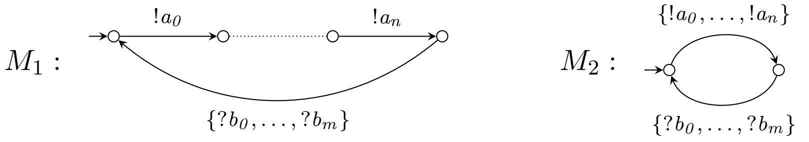
Implementation. The tool is implemented in Haskell and it mostly follows the structure of § 3. (1) It takes two machines M_1 and M_2 as input for which it builds a simulation tree following Definition 3.2 in a depth-first search manner, while recording the nodes visited in different branches to avoid re-computing several times the same subtrees. The function terminates whenever it expands a node whose label has been seen along the path from the root; or whenever it expands a node which has two ancestors that validate the termination condition from Theorem 3.8. The resulting tree is then passed onto the next function. (2) The next function divides the finite tree into several (finite) subtrees following the strategy outlined on page 16. (3) A third function analyses each subtree to verify that they validate conditions (1)-(2b) of Definition 3.16. (4) Finally, for those subtrees that validate the property checked in (3), the tool builds their systems of input tree equations and checks whether they validate the compatibility condition from Definition 3.12.

In function (1), if the tool finds a node for which none of the rules of Definition 3.2 apply, then it says that the two types are *not* related. If each subtree identified in (2) corresponds to branches that loop or that lead to a witness tree, then the tool says that the input types are in the subtyping relation. In all other cases, the result is still *unknown*, hence the tool checks for $\overline{M_2} \preccurlyeq \overline{M_1}$ (relying on a previous result showing that $M_1 \preccurlyeq M_2 \iff \overline{M_2} \preccurlyeq \overline{M_1}$ [LY17,BCZ17]). Once this pass terminates, the tool returns *true* or *false*, accordingly, otherwise the result is *unknown*.

For debugging and illustration purposes, the tool can optionally generate graphical representations of the simulation and candidate trees, as well as the systems of input tree equations.

Evaluation. We have run our tool on 174 tests which were either taken from the literature on asynchronous subtyping [LY17, CDSY17], or handcrafted to test the limits of our approach. All of these tests terminate under a second. Out of these tests, 92 are *negative* (the types are not in the subtyping relation) and our tool gives the expected result ("false") for all of them. The other 82 tests are *positive* (the types are in the subtyping relation) and our tool gives the expected result ("true") for all but 8 tests, for which it returns "unknown". All of these 8 examples feature complex accumulation patterns, that our theory cannot recognise. Example 3.21 gives a pair of machines for which our tool returns "unknown" for both $M_1 \preccurlyeq M_2$ and $\overline{M_2} \preccurlyeq \overline{M_1}$.

To assess the cost of our approach in terms of computation time and memory consumption, we have automatically generated a series of pairs of communicating machines that are successfully identified by our algorithm to be in the asynchronous subtyping relation. Our benchmarks consists in applying our algorithm to check that $M_1 \preccurlyeq M_2$ holds, with M_1 and M_2 as specified below, where $n, m \in \mathbb{N}_{>0}$ are the parameters of our experiments.



Machine M_1 sends a sequence of n message a_i, after which it expects to receive a message from the alphabet $\{b_0, \ldots, b_m\}$, then returns to its initial state. Machine M_2 can choose to send any message in $\{a_0, \ldots, a_n\}$, then waits for a message in $\{b_0, \ldots, b_m\}$ before returning to its initial state. Observe that for any n and m we have that $M_1 \preccurlyeq M_2$ holds. The shape of these machines allows us to assess how our approach fares in two interesting cases: when the sequence of message accumulation grows (i.e., n grows) and when the number of possible branches grows (i.e., m grows). Accordingly, we ran two series of benchmarks. The plots in Figure 7 gives the time taken for our tool to terminate and the maximum amount of memory used during its execution (left and right y axis, respectively) with respect to the parameter n (left-hand side plot) or m (right-hand side plot). The top plots use linear scales for both axes, while the bottom plots show the same data but using a logarithm scale for the y axis.

Observe that the left-hand side plot depicts a much steeper curve for computational time than the one of the right. Indeed, the depth of the finite subtree that needs to be computed and analysed increases with n (the depth of the finite subtree is $2n + 5$ when $m = 1$). Accordingly, the depth of the input contexts that need to be recorded increases similarly ($2n + 1$). Each input context node has two children in this case, i.e., $\langle b_0 : \langle \ldots \rangle, b_1 : \langle \ldots \rangle \rangle$.

In contrast, when m increases the depth of the simulation tree is bounded at 11. Consequently, the sizes of the finite subtrees are stable (depth of 7 when $n = 1$) but the number of (identical) candidate subtrees that need to be analysed increases, i.e., the tool produces m+1 trees when n=1. In this case the maximum depth of input contexts is also stable (the maximum depth is 3) but their widths increase with m, i.e., we have input context of the form: $\langle b_0 : \langle \ldots \rangle, \ldots, b_m : \langle \ldots \rangle \rangle$. These observations suggest that our algorithm is better suited to deal with session types that feature few anticipation steps (smaller n), but performs relatively well with types that contain many branches (larger m).

The left-hand side plots show that the memory consumption follows a similar exponential growth to the computational time, unsurprisingly. For instance, our tool needs 2GB to check a pair machines where $n = 10$ and $m = 1$, and 8 GB when $n = 11$ and $m = 1$. The right-hand side plots show a much smaller memory footprint when m increases, this is explained by the fact that the depth of the simulation tree is bounded, only the input context of its nodes are growing in width. The memory in this case is more reasonable, e.g., our tool needs less than 11MB to check a pair of machines where $n = 1$ and $m = 19$. We suspect the several jumps in the memory usage curve are due to the GHC runtime requesting new arenas of memory from the operating system.

All the benchmarks in this paper were run on an 8-core Intel i7-7700 machine with 16GB RAM running a 64-bit Linux. The time was measured by taking the difference between the system clock before and after our tool was invoked. The memory usage refers to the *maximum resident set size* as reported by the `/usr/bin/time -v` command. Each test was ran 5 times, the plots report the average time (resp. memory) measurements. All our test data and infrastructure are available on our GitHub repository [BCL+19b].

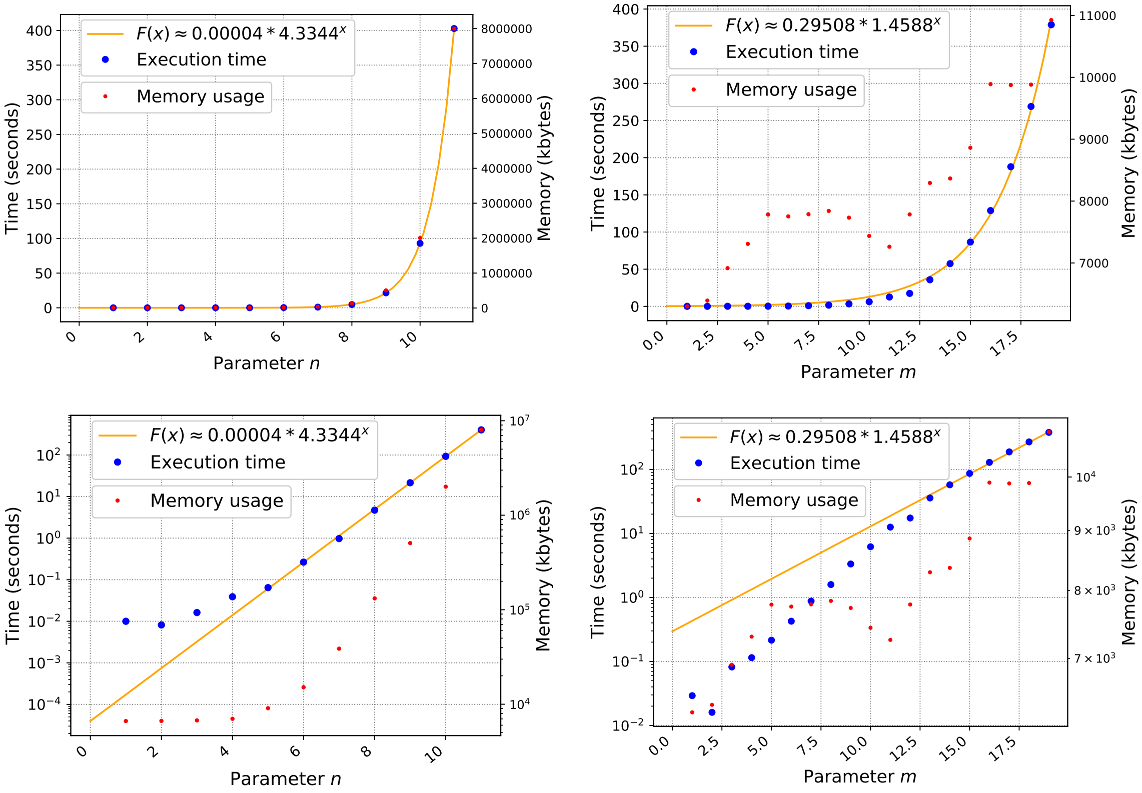


Figure 7: Benchmarks: m=1 and increasing n (left) and n=1 and increasing m (right). Top and bottom plot show the same data, but the top plots use linear scales for all axes, the bottom plots use logarithmic scales for the vertical axes.

5. Related Work

Gay and Hole [GH99, GH05] were the first to introduce subtyping for session types. Their definition, called *synchronous subtyping*, focuses on the possibility for a subtype to have different sets of labels in selections and branchings. In that paper, input selection is covariant (the subtype can have less inputs) while output branching is contravariant (the subtype can have more outputs). In our formulation of subtyping we have the opposite (branchings are covariant and selections are contravariant) because we follow a process-oriented interpretation of session types, while Gay and Hole [GH99, GH05] followed a channel-oriented interpretation.

Later, Mostrous et al. [MYH09] extended such notion to *asynchronous subtyping*, by allowing for delayed inputs. Chen et al. [CDCY14, CDSY17] subsequently provided an alternative definition which prohibits *orphan messages* and which is the definition we adopted in this work. Recently, asynchronous subtyping was shown to be undecidable by reducing it to an equivalent problem for Turing machines [LY17] and queue machines [BCZ17].

Our previous work [LY17, BCZ17, BCZ18] investigated different restrictions to achieve decidability: in all of our previous approaches, these restrictions are either (i) setting bounds on the number of pending messages in the FIFO channels, or (ii) restricting the syntax of communicating machines and session types. Lange and Yoshida [LY17, § 4] identified two subclasses of (two-party) communicating machines for which the asynchronous subtyping relation is decidable via syntactical restrictions: *alternating machines* and *non-branching machines*. Alternating machines were introduced by Gouda et al. [GMY84] and require that each sending transition is followed by a receiving transition. A consequence of this restriction is that each FIFO queue may contain at most one pending message, i.e., it enforces a form of 1-bounded asynchrony. Non-branching machines enforce a syntactical restriction such that each state has at most one outgoing transition, i.e., given $M = (Q, q_0, \delta)$, $\forall q \in Q \ : \ |\delta(q)| \leq 1$. Bravetti et al. [BCZ17, BCZ18] investigate other decidable fragments of asynchronous subtyping. In contrast with the present work and those by Lange and Yoshida, they take a direct syntactical approach, i.e., they work directly on the syntax of (binary) session types rather than communicating machines. Chronologically, their first article [BCZ17] proves the undecidability of asynchronous session subtyping in a restricted setting in which subtypes are non-branching (see definition above) in all output selections and supertypes are non-branching in all their input branchings. Then, a decidability result is proved for a fragment in which they additionally impose that the subtype is also non-branching in input branchings (or that the supertype is also non-branching in output selections). Later, in [BCZ18], the same authors consider more fragments, namely k-bounded asynchronous subtyping (bound on the size of input-anticipations), and two syntactical restrictions that imposes non-branching only on outputs (resp. inputs). More formally, following the automata notation, they restrict to machines M s.t. $M = (Q, q_0, \delta)$, $\forall q \in Q' \ : \ |\delta(q)| \leq 1$ with Q' coinciding with the set of sending (resp. receiving) states of Q. All such fragments are shown to be decidable.

In [BLZ21], Bravetti et al. propose a *fair* variant of asynchronous session subtyping. This fair subtyping handles candidate subtypes that may simultaneously send each other a finite but unspecified amount of messages before removing them from their respective buffers. Such types are not supported by the relation studied here, notably due the finiteness of input contexts $\mathcal{A}$ and the $\neg\mathsf{cycle}(!, p)$ condition in Definition 2.6 (3b). This fair subtyping is shown to be undecidable, but a sound algorithm and its implementation are given in [BLZ21].

The relationship between communicating machines and *binary* asynchronous session types has been studied in [BZ21], where a correspondence result between asynchronous session subtyping and asynchronous machine refinement is established. On the other hand, the relationship between communicating machines and *multiparty* asynchronous session types has been studied in [DY12, DY13]. Communicating machines are Turing-powerful, hence their properties are generally undecidable [BZ83]. Many variations have been introduced in order to recover decidability, e.g., using (existential or universal) bounds [GKM07], restricting to different types of topologies [LMP08, PP92], or using bag or lossy channels instead of FIFO queues [CHS14, CFI96, AJ93, ABJ98]. In this context, existentially bounded communicating machines [GKM07] are one of the most interesting sub-classes because they include infinite state systems. However, deciding whether communicating machines are existentially bounded is generally undecidable. Lange and Yoshida [LY19] proposed a (decidable) property that soundly characterises existential boundedness on communicating machines corresponding to session types. This property, called *k-multiparty compatibility* (k-MC), also guarantees that the machines validate the *safety* property of session types [DY13, LTY15], i.e., all messages that are sent are eventually received and no machine can get permanently stuck waiting for a message. This notion of safety is closely related to asynchronous session subtyping for two-party communicating machines, i.e., we have that $M_1 \preccurlyeq \overline{M_2}$ implies that the system $M_1 \mid M_2$ is safe [LY17, CDSY17]. Because the present work is restricted to two-party systems, our algorithm cannot be used to verify the safety of multiparty protocols, e.g., the protocol modelling the double-buffering algorithm [MYH09] is 2-multiparty compatible but cannot be verified with our subtyping algorithm because it involves three parties. This algorithm is used in multicore systems [SK08] and can be type-checked up-to asynchronous subtyping [MYH09]. An extension of our work to support multiparty protocols is being considered, see § 6. We note that because the k-MC property of [LY19] is based on a bounded analysis, it cannot guarantee the safety of systems that exhibit an intrinsically unbounded behaviour, like machines M_R and M_S in Figures 1 and 2.

6. Conclusions and Future Work

We have proposed a sound algorithm for checking asynchronous session subtyping, showing that it is still possible to decide whether two types are related for many nontrivial examples. Our algorithm is based on a (potentially infinite) tree representation of the coinductive definition of asynchronous subtyping; it checks for the presence of finite witnesses of infinite successful subtrees. We have provided an implementation and applied it to examples that cannot be recognised by previous approaches.

Although the (worst-case) complexity of our algorithm is rather high (the termination condition expects to encounter a set of states already encountered, of which there may be exponentially many), our implementation shows that it actually terminates under a second for machines of size comparable to typical communication protocols used in real programs, e.g., Go programs feature between three and four communication primitives per channel and whose branching construct feature two branches, on average [DL19].

As future work, we plan to enrich our algorithm to recognise subtypes featuring more complex accumulation patterns, e.g., Example 3.21. Moreover, due to the tight correspondence with safety of communicating machines [LY17], we plan to investigate the possibility of using our approach to characterise a novel decidable subclass of communicating machines. It is an interesting open question to extend our algorithm to multiparty communications, as multiparty session types allow more permutations of actions inside a single CFSM and can type more practical use cases which involve several participants. Recently *precise* multiparty asynchronous subtyping (in the sense of [CDCY14, CDSY17, GJP$^+$19]) for the asynchronous multiparty session π-calculus [HYC08, HYC16] was proposed in [GPP$^+$21]. In another direction of future work we will consider an algorithm for checking subtyping which is sound, but not complete with respect to [GPP$^+$21]. Finally, a significant further extension could be to also encompass pre-emption mechanisms, see e.g. [BZ09, Bra21], which are often used in communication protocols.

Acknowledgement

The work is partially funded by H2020-MSCA-RISE Project 778233 (BEHAPI); and EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, EP/N028201/1, EP/T014709/1, EP/V000462/1, EP/T006544/1 and NCSC VeTSS.

References

[ABB$^+$16] Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016.

[ABJ98] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *CAV 1998*, pages 305–318, 1998.

[AJ93] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *(LICS 1993)*, pages 160–170, 1993.

[BCL$^+$19a] Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. A sound algorithm for asynchronous session subtyping. In *CONCUR*, volume 140 of *LIPIcs*, pages 38:1–38:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[BCL$^+$19b] Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. A sound algorithm for asynchronous session subtyping. `https://github.com/julien-lange/asynchronous-subtyping`, 2019.

[BCZ17] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. Undecidability of asynchronous session subtyping. *Inf. Comput.*, 256:300–320, 2017.

[BCZ18] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. On the boundary between decidability and undecidability of asynchronous session subtyping. *Theor. Comput. Sci.*, 722:19–51, 2018.

[BLZ21] Mario Bravetti, Julien Lange, and Gianluigi Zavattaro. Fair refinement for asynchronous session types. In *FoSSaCS*, Lecture Notes in Computer Science, 2021. To appear. Available at `https://arxiv.org/abs/2101.08181`.

[Bra21] Mario Bravetti. Axiomatizing maximal progress and discrete time. *Log. Methods Comput. Sci.*, 17(1), 2021.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[BZ09] Mario Bravetti and Gianluigi Zavattaro. On the expressive power of process interruption and compensation. *Math. Struct. Comput. Sci.*, 19(3):565–599, 2009.

[BZ20] Mario Bravetti and Gianluigi Zavattaro. Process calculi as a tool for studying coordination, contracts and session types. *J. Log. Algebraic Methods Program.*, 112:100527, 2020.

[BZ21] Mario Bravetti and Gianluigi Zavattaro. Asynchronous session subtyping as communicating automata refinement. *Software and Systems Modeling*, 2021. `https://doi.org/10.1007/s10270-020-00838-x`.

[CDCY14] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types. In *PPDP 2014*, pages 146–135. ACM Press, 2014.

[CDSY17] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the preciseness of subtyping in session types. *Logical Methods in Computer Science*, 13(2), 2017.

[CFI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996.

[CHS14] Lorenzo Clemente, Frédéric Herbreteau, and Grégoire Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In *CONCUR 2014*, pages 281–296, 2014.

[dBBLZ18] Frank S. de Boer, Mario Bravetti, Matias David Lee, and Gianluigi Zavattaro. A petri net based modeling of active objects and futures. *Fundam. Informaticae*, 159(3):197–256, 2018.

[DL19] N. Dilley and J. Lange. An empirical study of messaging passing concurrency in Go projects. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 377–387, Feb 2019.

[DY12] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP 2012*, pages 194–213, 2012.

[DY13] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP 2013*, pages 174–186, 2013.

[GH99] Simon J. Gay and Malcolm Hole. Types and subtypes for client-server interactions. In *ESOP 1999*, pages 74–90, 1999.

[GH05] Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005.

[GJP$^+$19] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *JLAMP*, 104:127–173, 2019.

[GKM07] Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007.

[GMY84] Mohamed G. Gouda, Eric G. Manning, and Yao-Tin Yu. On the progress of communications between two finite state machines. *Information and Control*, 63(3):200–216, 1984.

[GPP$^+$21] Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, and Nobuko Yoshida. Precise Subtyping for Asynchronous Multiparty Sessions. 5:16:1–16:28, 2021. A full version is available from `https://arxiv.org/abs/2010.13925`.

[HY16] Raymond Hu and Nobuko Yoshida. Hybrid session verification through endpoint API generation. In *FASE 2016*, pages 401–418, 2016.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL'08*, pages 273–284. ACM, 2008.

[HYC16] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016.

[JM99] Petr Jancar and Faron Moller. Techniques for decidability and undecidability of bisimilarity. In *CONCUR 1999*, pages 30–45, 1999.

[LM16] Sam Lindley and J. Garrett Morris. Embedding session types in Haskell. In *Haskell 2016*, pages 133–145, 2016.

[LMP08] Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS 2008*, pages 299–314, 2008.

[LTY15] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL 2015*, pages 221–232, 2015.

[LY16] Julien Lange and Nobuko Yoshida. Characteristic formulae for session types. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 833–850. Springer, 2016.

[LY17] Julien Lange and Nobuko Yoshida. On the undecidability of asynchronous session subtyping. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 441–457, 2017.

[LY19] Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. In *Computer Aided Verification - 31st International Conference*, volume 11561 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2019.

[MY15] Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015.

[MYH09] Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP 2009*, pages 316–332, 2009.

[NHYA18] Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F♯. In *CC 2018*. ACM, 2018.

[OY16] Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *POPL 2016*, pages 568–581, 2016.

[Pad17] Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017.

[PP92] Wuxu Peng and S. Purushothaman. Analysis of a class of communicating finite state machines. *Acta Inf.*, 29(6/7):499–522, 1992.

[SK08] Jose Sancho and Darren Kerbyson. Analysis of double buffering on two different multicore architectures: Quad-core opteron and the cell-be. pages 1–12, 04 2008.

[SY16] Alceste Scalas and Nobuko Yoshida. Lightweight session programming in scala. In *ECOOP 2016*, pages 21:1–21:28, 2016.